VISUAL BASIC 6 FOR DUMMIES

NOVITÀ

Espresso

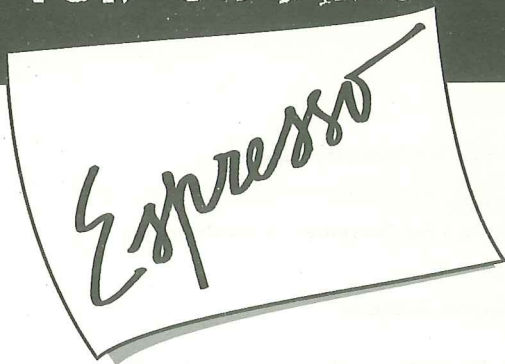
Namir Clement Shammas

I piccoli manuali per comprendere l'informatica

Una lettura veloce per cogliere l'essenza di Visual Basic 6

Tutti i comandi e tutte le funzioni in poche parole

Un manuale facile, intelligente ed economico

VISUAL BASIC 6 FOR DUMMIES

Namir Clement Shammas

APOGEO

Visual Basic 6 For Dummies® Espresso

Titolo originale:
Visual Basic 6 For Dummies® Quick Reference

Autore:
Namir Clement Shammas

Viale Papiniano 38 – 20123 Milano (Italy)
Telefono: 02-461920 (5 linee r.a.) – Telefax: 02-4815382
Email **apogeo@apogeonline.com**
U.R.L. **www.apogeonline.com**

ISBN 88-7303-476-4

Revisione tecnica di Piemme Informatica
Impaginazione elettronica di Datapress srl – Milano

Sommario

Come usare questa guida

Questo libro vuole essere un riferimento rapido per tutti i comandi, le procedure e gli elementi del codice di Visual Basic. Data questa premessa, non si pensi di trovare qui quelle informazioni altamente specifiche per le procedure o i comandi che verranno utilizzati molto raramente.

Questo non è un libro da leggere sequenzialmente dalla prima all'ultima pagina (non è un romanzo) e, pertanto, per reperire un argomento specifico si faccia riferimento al sommario, al glossario finale o all'indice analitico. Se non si è sicuri di quale posizione di codice si ha necessità, si passi direttamente alla sezione che si presume tratti dell'argomento desiderato e se ne scorra il contenuto. Lo scopo finale di questa guida è quello di assistere e aiutare l'utente nella creazione dei propri programmi e di chiarire rapidamente i concetti della programmazione in Visual Basic. In questo modo si potrà subito procedere alla creazione dei propri programmi.

Contenuto del libro

Questa guida è stata suddivisa in undici sezioni tematiche, nelle quali si potranno reperire rapidamente gli argomenti di cui si ha necessità.

- **Parte I** - *Concetti fondamentali di Visual Basic.* Vengono qui fornite tutte le informazioni per operare con l'ambiente di programmazione di Visual Basic 6, per "costruire" un'applicazione, per usare le varie finestre e, in linea generale, come interagire con l'interfaccia Visual Basic. In questa parte vengono anche introdotti alcuni concetti e termini essenziali (quali, per esempio, l'enunciato Print) che si ritroveranno anche nelle altre pagine del libro.
- **Parte II** - *Controlli semplici.* Questa parte è dedicata a una descrizione dei controlli di tipo più semplice, il cui scopo è quello di consentire, all'utilizzatore finale dell'applicazione, di inserire, per esempio, propri dati o di visualizzare elementi testuali.
- **Parte III** - *Classi e tipi di dati.* Si descrivono qui i moduli di classe (.CLS) e i vari tipi di dati messi a disposizione da Visual Basic e si indica come crearne di propri. Si analizzeranno, inoltre, i vari operatori che consentiranno di confrontare i diversi elementi di Visual Basic.
- **Parte IV** - *Le finestre di dialogo.* Questa parte è dedicata alla descrizione delle varie finestre di dialogo che potranno essere create per facilitare l'interazione fra il programma che viene "costruito" e l'utente che dovrà utilizzarlo.
- **Parte V** - *Controlli avanzati.* Contiene le informazioni relative ai controlli di tipo avanzato, alcuni dei quali sono disponibili solo nelle edizioni "Professional" e "Enterprise" di Visual Basic 6.
- **Parte VI** - *Gestione degli errori.* Si tratta di un riferimento rapido su come indicare al programma di prendere le proprie decisioni e su come gestire eventuali errori.
- **Parte VII** - *Funzioni e procedure.* Questa parte contiene tutto ciò che è necessario sapere per inserire piccoli programmi all'interno di programmi più complessi.
- **Parte VIII** - *Manipolazione delle stringhe.* Si spiega come usare ed estrarre parti di stringhe di testo.
- **Parte IX** - *Gestione di form e MDI.* In questa parte vengono presi in considerazione i form e l'MDI (Multiple Document Interface).

- **Parte X** - *Manipolazione e accesso alle matrici (array).* Si prendono in esame le matrici multidimensionali e si descrive come inserirvi o richiamarne dati.

- **Parte XI** - *Uso dei cicli.* Questa parte finale si occupa della creazione di cicli per l'iterazione dei compiti richiesti a un programma e si indica come sia possibile interrompere un ciclo nel caso in cui venga soddisfatta una condizione particolare.

- **Glossario.** Contiene un elenco di termini tecnici, associati a un breve testo descrittivo, correlati all'uso di Visual Basic 6.

Convenzioni adottate

Per facilitare la lettura delle informazioni riportate nella guida, sono state adottate alcune convenzioni.

Quando viene richiesto di scrivere qualcosa all'interno di un passo di una procedura numerata, il testo da digitare verrà riportato in **grassetto**. Quanto indicato dovrà essere scritto esattamente come viene proposto. In varie occasioni vengono proposte sequenze procedurali numerate che contengono i passi sequenziali da intraprendere per eseguire quanto richiesto.

1. I passi delle sequenze procedurali vengono sempre inseriti in elenchi numerati.
2. Si raccomanda di eseguire i passi della procedura nell'ordine proposto.
3. Si concluda la procedura sequenziale come proposto.

In altre occasioni, invece, vengono utilizzati elenchi puntati, simili a quello seguente.

- Alcune di queste procedure prevedono operazioni che potranno essere eseguite in un ordine qualsiasi.

- Altre volte gli elenchi puntati riportano solo una serie di informazioni descrittive.

Le linee di codice vengono sempre proposte con un carattere a spaziatura fissa diverso dal testo corrente, assumendo un aspetto simile al seguente.

```
[You | I] Love NomeCosa
   nomeElemento [([persona])] As tipo
   [nomeElemento [([persona])] As tipo]
```

```
...
End Love
```

Ciò che nel codice viene riportato in **grassetto** rappresenta quanto dovrà essere scritto (esattamente come viene proposto), a meno che, come nel caso di `[You ¦ I]` della prima riga dell'esempio, la richiesta di digitazione venga separata dal simbolo |. In questo caso, si ha una possibilità di scelta fra le alternative proposte. Le varie voci del codice non sono altro che nomi che il programmatore dovrà assegnare ai corrispondenti elementi del codice. Sempre prendendo in considerazione l'esempio sopra proposto, `NomeCosa` potrebbe essere sostituito da `Frutta`. La parte di codice racchiusa fra parentesi quadre, `[ ]`, è da considerarsi facoltativa.

Spesso, la sintassi di una voce è molto più chiara dopo che la si sarà inserita in un esempio reale e, pertanto, non si abbia alcuna esitazione nell'utilizzare i vari esempi proposti, anche se questi, a prima vista, non appaiono immediatamente comprensibili.

La maggior parte delle voci descritte nel corso del libro vengono sempre associate a una breve descrizione della loro operatività, alla sintassi corrispondente e a un esempio pratico. Molte voci, inoltre, prevedono la presenza di tabelle che riassumono gli argomenti ammissibili. Alcuni dei programmi di esempio vengono accompagnati da figure in modo da mostrare esattamente cosa ci si deve aspettare.

Infine, l'indicazione ***Vedi anche:*** rimanda a una particolare sezione o paragrafo del libro. È una sorta di riferimento incrociato per integrare quanto descritto nelle righe di testo precedenti.

Quando viene riportato un indirizzo Internet o un URL, questo apparirà in carattere a spaziatura fissa (per esempio: `http://www.dummies.com`) e, inoltre, quando questo appare alla fine di una frase o di un paragrafo, in base alle regole di composizione, verrà seguito da un punto (.) conclusivo della frase o del paragrafo stessi. Si raccomanda, in questo caso, di scrivere l'indirizzo senza il punto finale.

La sequenza di selezione di un comando da menu viene riportata con il nome del menu (completo di lettera sottolineata) seguito dal simbolo ⇨ e, quindi, dal comando richiesto. Per esempio, l'indicazione File ⇨ Salva col nome richiede l'apertura del menu File e la successiva attivazione del comando Salva col nome. Se un comando genera l'apertura di un sottomenu per selezionare una voce ulteriore, anche quest'ultima verrà preceduta dal simbolo ⇨ (per esempio: File ⇨ Nuovo ⇨ Browser).

Alcuni tasti vengono rappresentati graficamente, in conformità ad altre guide della stessa collana.

- ← indica Backspace ovvero il tasto posto nell'angolo superiore destro della parte principale della tastiera.
- ↵ indica Invio.
- ↑ → ↓ ← indicano i tasti per lo spostamento del cursore (o tasti freccia posti in basso tra la tastiera principale e il tastierino numerico). Una copia di questi tasti si trova anche sul tastierino numerico.
- ⇨ (Maiusc) indica il tasto che deve essere premuto e mantenuto tale per inserire caratteri maiuscoli.
- Pag ↑ (Pag Su) indica il tasto corrispondente posto tra la tastiera principale e il tastierino numerico (che contiene, tra l'altro, una copia del tasto).
- Pag ↓ (Pag Giù) indica il tasto corrispondente posto tra la tastiera principale e il tastierino numerico (che contiene, tra l'altro, una copia di tale tasto).

Quando è necessario premere e mantenere premuto un tasto e successivamente premerne un secondo (e in alcuni casi un terzo) tale combinazione viene riportata indicando il primo tasto, un segno "+" e quindi il tasto successivo. Per esempio, la notazione Ctrl+N indica di premere e mantenere premuto il tasto Ctrl, premere il tasto "N" e rilasciare il tasto Ctrl.

Il mouse è dotato, in genere, di due pulsanti: uno primario (di solito identificato con quello sinistro) e uno secondario (di solito, quello destro). Quando si richiede di fare clic, si intende premere e rilasciare rapidamente il pulsante primario del mouse. Si utilizzi il pulsante secondario solo quando indicato, poiché il suo uso attiva alcune funzioni particolari (per esempio, la comparsa dei menu contestuali).

I messaggi emessi dall'applicazione e visualizzati sullo schermo vengono riportati utilizzando un carattere a spaziatura fissa, il carattere `MCPdigital`.

Icone utilizzate

Tutti i libri che trattano di computer e applicazioni in genere prevedono ormai l'uso di una serie di icone (piccole immagini a corredo del testo) per richiamare l'attenzione del lettore su paragrafi particolari. Questa guida non si discosta da tale filosofia e, pertanto, è bene spiegare il significato delle icone utilizzate.

Quanto viene associato a questa icona rappresenta e descrive l'uso di un elemento Visual Basic.

Con questa icona si contrassegnano gli esempi di elementi Visual Basic che possono essere utilizzati per la creazione di un programma reale.

Con questa icona si segnala un paragrafo che contiene un suggerimento per eseguire un'operazione (di solito un'alternativa a quanto si sta eseguendo).

I paragrafi associati a questa icona contengono informazioni da valutare con particolare attenzione poiché potrebbero generare situazioni di tipo distruttivo!

Parte I

Concetti fondamentali di Visual Basic

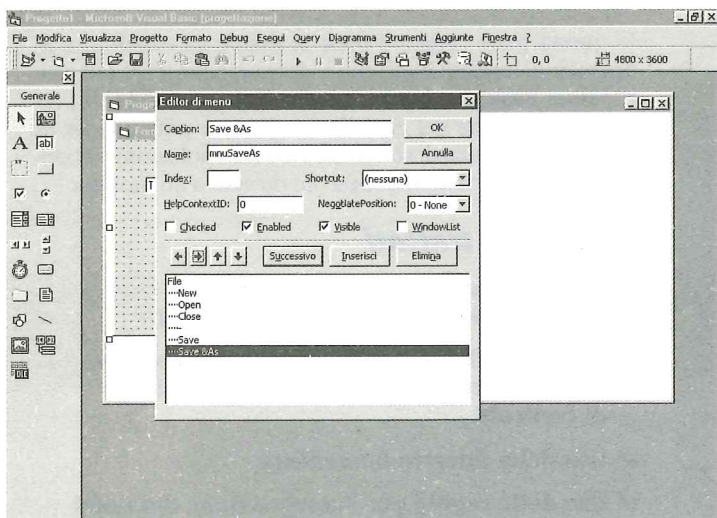
In questa parte vengono presi in esame gli elementi e le operazioni fondamentali che si devono conoscere per usare Visual Basic. Sia che si conosca già l'ambiente di programmazione, sia nel caso in cui si sia un principiante, in queste pagine iniziali si troveranno molte soluzioni per evitare di dover rimanere troppe ore davanti a un monitor.

Argomenti trattati

- ✔ **Inserimento, spostamento, modifica ed eliminazione di menu e sottomenu**
- ✔ **Disegno, allineamento, spostamento, copia, incolla, eliminazione e personalizzazione di controlli**
- ✔ **Uso della finestra *Immediata***
- ✔ **Uso delle regole per l'assegnazione dei nomi in Visual Basic**
- ✔ **Integrazione con Visual Studio**
- ✔ **Le finestre *Progetto* e *Proprietà***
- ✔ **Operazioni sui file di progetto**

Aggiunta di menu

L'inserimento di un menu in un form è un'operazione particolarmente facile: si proceda come segue.

1. Si faccia clic con il pulsante destro del mouse all'interno del form. Viene aperto un menu di scelta rapida.
2. Si faccia clic su Menu Editor.

Nei paragrafi successivi verrà descritto più dettagliatamente come sia possibile utilizzare la finestra Menu Editor per eseguire le operazioni più comuni.

Ricordare. Un *nome di menu* (o *menu*) viene visualizzato all'interno della Barra dei menu che si sviluppa orizzontalmente immediatamente sotto la Barra del titolo della finestra. Un *comando di menu* (o *voce di menu*) *è* una delle voci che vengono visualizzate quando il menu viene aperto.

Creazione di una nuova voce di menu

Per creare un nuovo menu si proceda come segue.

1. Nella casella Caption si scriva il testo da associare alla voce di menu che si desidera creare. Mentre si scrive la nuova voce, si inserisca il simbolo "&" prima del carattere che si desidera appaia sottolineato, per identificarlo come carattere per l'apertura rapida del menu stesso. Per esempio, se nel-

la casella si scrivesse `&File` la voce di menu verrebbe riportata come File, consentendo, così, di aprire il menu con la semplice pressione di Alt+F.

2. Nella casella di testo Name si scriva il nome da assegnare alla voce di menu. Di solito, le convenzioni prevedono l'indicazione del nome del menu preceduto dalle lettere `mnu`.

Per aggiungere altre voci di menu, si faccia clic sul pulsante di comando Successivo o su quello Inserisci e si ripeta la procedura sopra riportata.

Creazione di un comando di un menu

Per inserire il primo menu, si proceda come segue.

1. Si faccia clic sul pulsante che riporta una freccia rivolta verso destra (quello posto sotto la casella di controllo dell'opzione Checked) per "scendere di un livello" nella gerarchia di creazione dei menu e dei relativi comandi.
2. Nella casella Caption si scriva il testo che identifica il comando che si sta creando, utilizzando, prima della lettera che dovrà apparire sottolineata, il simbolo "&". Per esempio, se si scrivesse `&Esci`, il comando creato assumerebbe la forma Esci, consentendo così all'utente di premere semplicemente il tasto "E" per eseguire il comando.
3. Nella casella Name si scriva il nome del comando di menu. Di solito, per convenzione, tale nome viene sempre preceduto dalle tre lettere `mnu`.

Per creare altri comandi di un menu, si faccia clic sul pulsante di comando Successivo o su quello Inserisci e si esegua la procedura sopra indicata.

- Per inserire in un menu una riga separatrice per i vari gruppi di comandi nella casella Caption si inserisca il carattere "-" e nella casella Name si scriva, per esempio, N1.
- Utilizzare i pulsanti contenenti le frecce rivolte verso destra o verso sinistra per salire o scendere all'interno della "gerarchia" dei livelli di creazione di menu. L'uso dei pulsanti contenenti le freccie rivolte verso l'alto o verso il basso consente, invece, di modificare la posizione di un menu o di un suo comando.

Creazione di sottomenu

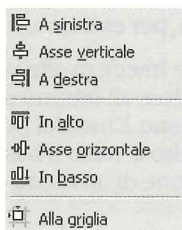
Per creare un sottomenu si proceda come segue.

1. Si faccia clic sul pulsante con la freccia rivolta verso destra (posto sotto l'opzione Checked) per scendere di un livello nella gerarchia dei menu, accedendo, così, a quello per la creazione dei sottomenu.
2. Nella casella di testo Caption si scriva il nome da assegnare al sottomenu, facendo precedere dal simbolo "&" il carattere che dovrà apparire sottolineato. Per esempio, se si scrivesse `&Calcola`, verrebbe visualizzato Calcola, consentendo così all'utente di premere semplicemente "C" per accedere ai comandi del sottomenu stesso.
3. Nella casella Name si scriva il nome da associare al sottomenu. Di solito, per convenzione, tale nome viene sempre preceduto da `mnu`.

Per creare altri nomi di sottomenu, si faccia clic sul pulsante di comando Successivo o su quello Inserisci e si esegua di nuovo la procedura sopra indicata.

Allineamento di più controlli

Per allineare contemporaneamente due o più controlli di un form si proceda come segue.

1. Si selezionino i controlli che si desiderano allineare facendo clic su ognuno di essi premendo e mantenendo premuto il tasto ⇧ (Maiusc).
2. Si scelga il comando Formato⇨Allinea per accedere al sottomenu corrispondente.

La tabella seguente descrive schematicamente i comandi contenuti nel menu Allinea.

Comando	Descrizione
A sinistra	Allinea orizzontalmente i controlli selezionati, posizionando quello più a sinistra in linea con il bordo verticale sinistro dell'ultimo controllo selezionato (ovvero di quello con le maniglie di controllo di colore nero pieno).
Asse verticale	Allinea orizzontalmente i controlli selezionati in base a un ideale asse verticale centrale che attraversa l'ultimo controllo selezionato (quello con le maniglie di controllo nero pieno).
A destra	Allinea orizzontalmente i controlli selezionati in base al bordo verticale destro dell'ultimo controllo selezionato (quello con le maniglie di controllo nero pieno).
In alto	Allinea verticalmente i controlli selezionati in base al bordo orizzontale superiore dell'ultimo controllo selezionato (quello con le maniglie di controllo nero pieno).
Asse orizzontale	Allinea verticalmente i controlli selezionati in base a un ideale asse orizzontale centrale che attraversa l'ultimo controllo selezionato (quello con le maniglie di controllo nero pieno).
In basso	Allinea verticalmente i controlli selezionati in base al bordo orizzontale inferiore dell'ultimo controllo selezionato (quello con le maniglie di controllo nero pieno).
Alla griglia	Allinea l'angolo superiore sinistro della serie di controlli selezionati al più vicino punto di griglia senza influire sulla dimensione originaria dei controlli stessi.

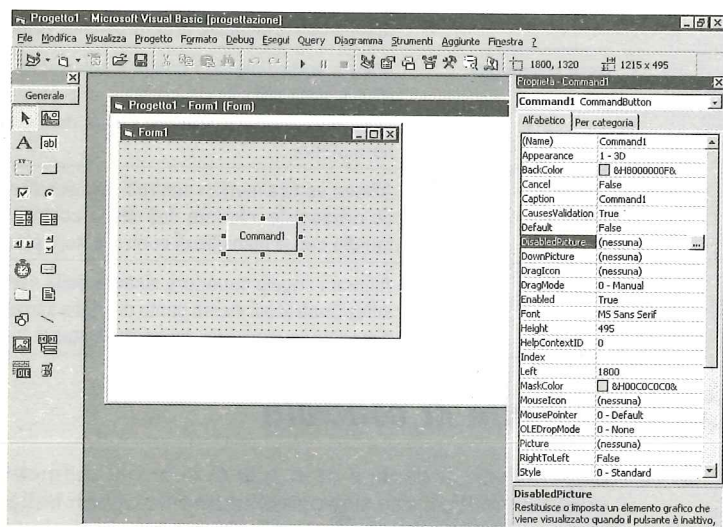
Copia e incolla di controlli

L'operazione di copia e di incolla di controlli permette di creare controlli multipli che siano assolutamente uguali nella dimensione e nell'aspetto (con la sola eccezione eventualmente della proprietà `Name`). Dopo una duplicazione di uno o più controlli questi potranno essere conservati nella loro forma e dimensione originale oppure sarà possibile intervenire su alcune delle proprietà corrispondenti. Per copiare e incollare (quindi duplicare) un controllo, si proceda come segue.

1. Si selezioni il controllo da duplicare facendovi sopra clic.
2. Si selezionino (se necessario) gli altri controlli da duplicare contemporaneamente al primo selezionato, facendovi sopra clic mantenendo premuto il tasto ⇧ (Maiusc). Per deselezionare un controllo erroneamente selezionato, sempre mantenendo premuto il tasto ⇧ (Maiusc) vi si faccia sopra clic.

3. Si prema Ctrl+C
4. Si prema Ctrl+V
5. Si intervenga sulle proprietà (per esempio `Caption` oppure `Name`) dei controlli incollati per modificarle rispetto a quelle dei controlli originari.

Personalizzazione delle proprietà dei controlli

Visual Basic prevede, per ogni controllo, una propria finestra delle proprietà. La figura seguente presenta, per esempio, la finestra Proprietà alla quale si può accedere tramite il comando Visualizza ⇨ Finestra Proprietà.

La finestra delle proprietà di un controllo è composta da due sezioni che permettono di accedere a due tipi di visualizzazione delle proprietà del controllo: Alfabetico e Per categoria. La prima, elenca le proprietà in ordine alfabetico, mentre la seconda le elenca in base alle categorie di appartenenza. Le proprietà vengono visualizzate in una tabella su due colonne dove quella di sinistra contiene il nome della proprietà mentre quella destra ne indica il valore. Per modificare quindi il valore di una proprietà, si faccia clic con il mouse all'interno della casella che ne contiene il valore.

Visual Basic riconosce i seguenti tipi di proprietà.

- **Modificabile liberamente.** Questo tipo di proprietà (applicabile, per esempio, a un valore testuale o numerico intero) può essere modificato digitando nella casella corrispondente il nuovo valore oppure intervenendo su quello già presente. Per esempio, si potrà intervenire sulla proprietà `Caption` di un pulsante di comando.
- **A scelta da menu.** Quando si seleziona il valore di una proprietà di tipo enumerato, Visual Basic visualizza, alla destra del valore stesso, un pulsante contenente una freccia rivolta verso il basso. Se si fa clic su tale pulsante verrà visualizzato un elenco di valori predefiniti ammissibili. Un esempio tipico di queste proprietà è quello che prevede valori di tipo booleano (tanto per intenderci: `Enabled` oppure `Visible`) o proprietà che intervengono sul colore del controllo (`ForeColor` oppure `BackColor`).
- **A scelta guidata.** Quando si seleziona il valore di una di queste proprietà Visual Basic visualizza, alla destra del valore, un pulsante contenente tre puntini di sospensione (`...`). Facendo clic su tale pulsante si accederà a una finestra di dialogo che consentirà di selezionare un nuovo valore. Un esempio tipico di proprietà che utilizzano questo tipo di valori sono quelle che definiscono un nome di file o un'immagine.

Per maggiori informazioni circa la personalizzazione di un controllo, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Funzioni per le date

Visual Basic è dotato di un'ampia gamma di funzioni per la gestione delle date che verranno prese in esame nei paragrafi di questa sezione.

Vedi anche: in questa parte, "Funzioni per la gestione dell'ora".

La funzione Date( )

La funzione `Date( )` restituisce un `Variant` (`Date`) che corrisponde alla data di sistema. La sintassi generale della funzione `Date( )` è la seguente.

```
Date
```

Quello che segue è un esempio classico dell'uso della funzione.

```
Debug.Print Date ' visualizza la data di sistema
```

Questo esempio genera la visualizzazione, nella finestra Immediata, della data di sistema.

La funzione Day( )

La funzione `Day( )` restituisce un `Variant(Integer)`, un valore intero compreso tra 1 e 31, che rappresenta il giorno (del mese) di `Data`. `Data` può essere un `Variant`, un'espressione numerica, un'espressione di tipo stringa oppure una qualsiasi combinazione che possa rappresentare una data.

La sintassi generale della funzione `Day()` è la seguente.

Day(Data)

Viene ora proposto un esempio dell'uso della funzione `Date( )`

```
Dim miaData As Date
miaData = #5/3/98#
Debug.Print Day(miaData)
```

Con questo esempio viene dichiarata la variabile `miaData` di tipo `Date` alla quale è assegnata la data del 3 maggio 1998. L'enunciato di output richiama la funzione `Day( )` il cui argomento corrisponde a `miaData`. L'enunciato genera la visualizzazione, nella finestra Immediata, del numero 3.

La funzione Month( )

La funzione `Month( )` restituisce un `Variant(Integer)`, un numero intero compreso tra 1 e 12, che rappresenta la componente del mese di `Data`. `Data` può essere un `Variant`, un'espressione numerica, un'espressione di tipo stringa o una qualsiasi combinazione che rappresenti una data.

La sintassi generale della funzione `Month()` è la seguente.

Month(Data)

Viene ora presentato un esempio dell'uso della funzione `Month( )`.

```
Dim miaData As Date
miaData = #5/3/98#
Debug.Print Month(miaData)
```

Con questo esempio si dichiara la variabile `miaData` di tipo `Date` e le si assegna la data del 3 maggio 1998. L'enunciato di output richiama la funzione `Month( )` il cui argomento corrisponde alla variabile `miaData`. L'enunciato genera la visualizzazione, nella finestra Immediata, del numero 5.

La funzione Now( )

La funzione `Now( )` restituisce la data e l'ora di sistema come `Variant(Date)`.

La sintassi generale per la funzione `Now()` è la seguente.

```
Now
```

Qui di seguito viene proposto un esempio dell'uso della funzione

`Now( )`.

```
Debug.Print Now ' visualizza la data/ora di sistema attiva
```

Questo esempio genera la visualizzazione, nella finestra Immediata dell'ora e della data di sistema.

La funzione Weekday( )

La funzione `Weekday( )` restituisce il numero del giorno della settimana per un `Variant`, valore numerico, valore di tipo stringa o di una qualsiasi combinazione che rappresenti una data.

La sintassi generale per la funzione `Weekday( )` è la seguente.

```
Weekday(Data, [firstDayOfWeek])
```

La funzione `Weekday( )` restituisce un `Variant(Integer)` che specifica un numero intero che rappresenta un giorno della settimana per `Date`. L'argomento facoltativo `firstDayOfWeek` definisce il primo giorno della settimana e, se non altrimenti specificato, Visual Basic presume che la settimana inizi sempre di domenica.

La funzione `Weekday( )`, per l'argomento `firstDayOfWeek`, fa uso delle costanti riportate nella tabella seguente.

Costante	*Valore*	*Descrizione*
`vbUseSystem`	0	Usa le impostazioni NLS API
`vbSunday`	1	Domenica (default)
`vbMonday`	2	Lunedì
`vbTuesday`	3	Martedì
`vbWednesday`	4	Mercoledì
`vbThursday`	5	Giovedì
`vbFriday`	6	Venerdì
`vbSaturday`	7	Sabato

Viene ora proposto un esempio dell'uso della funzione `Weekday( )`

```
Dim miaData As Date
miaData = #5/3/98#
Debug.Print Weekday(miaData)
```

Questo esempio dichiara la variabile `miaData` di tipo `Date` assegnandole la data del 3 maggio 1998. L'enunciato di output richiama la funzione `Weekday( )` assegnandole l'argomento `miaData`. L'enunciato genera la visualizzazione, nella finestra Immediata, del numero 7 (corrispondente a sabato).

La funzione Year( )

La funzione `Year( )` restituisce il numero dell'anno per un particolare `Variant`, valore numerico, stringa o per qualsiasi espressione combinata che rappresenti una data.

La sintassi generale della funzione `Year()` è la seguente.

```
Year(Data)
```

La funzione restituisce un `Variant(Integer)` che definisce un numero intero che rappresenta la componente dell'anno di `Data`.

Quello che segue è un esempio dell'uso della funzione

```
Dim miaData As Date
miaData = #5/3/98#
Debug.Print Year(miaData)
```

Con questo esempio viene dichiarata la variabile `miaData` di tipo `Date` e le si assegna la data del 3 maggio 1998. L'enunciato di output richiama la funzione `Year( )` il cui argomento corrisponde alla variabile `miaData`. Il risultato dell'enunciato è la visualizzazione, nella finestra Immediata, del numero 1998.

Eliminazione di controlli

Per eliminare un controllo da un form, si proceda come segue.

1. Si selezioni il controllo che si desidera eliminare facendovi sopra clic. Se si desiderano eliminare più controlli contemporaneamente, si faccia clic su quelli da selezionare mantenendo premuto il tasto ⇧(Maiusc).
2. Si prema il tasto Canc, oppure si scelga il comando Modifica ⇨ Taglia.

Eliminazione di menu

Per eliminare un intero menu si richiami l'editor di menu cliccando con il tasto destro del mouse e selezionando la voce Menu Editor, quindi si proceda come segue.

1. Si selezioni il nome del menu o del comando che si desidera eliminare.
2. Si faccia clic sul pulsante Elimina.

Disegno dei controlli

Il disegno di vari controlli di un'applicazione è uno dei lati più piacevoli di Visual Basic poiché il loro utilizzo determina l'interfaccia dell'applicazione che si sta creando servendosi, per questa operazione, della finestra del form e della Casella degli strumenti. Per disegnare un controllo all'interno di un modulo, si proceda come segue.

1. Fare clic, nella Casella degli strumenti, sul controllo che si desidera creare.
2. Si porti il puntatore del mouse all'interno del form in modo da definire l'angolo superiore sinistro del controllo che si sta creando.
3. Si prema e si mantenga premuto il pulsante del mouse e si trascini il puntatore diagonalmente verso il basso e a destra per creare l'area nella quale dovrà apparire il controllo. Al puntatore viene agganciato un riquadro tratteggiato che indica la dimensione raggiunta dell'area del controllo che si sta creando.
4. Quando si sarà raggiunta la dimensione desiderata, si rilasci il pulsante del mouse. Visual Basic visualizza il controllo all'interno dello spazio definito assegnando al controllo stesso le proprietà di default.

Per maggiori informazioni sul disegno dei controlli, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Menu

Per questi elementi si rinvia alle sezioni di questa parte che si occupano specificatamente dell'aggiunta, dell'eliminazione, dello spostamento e della creazione di menu e sottomenu.

Spostamento di controlli

Per spostare un controllo si proceda come segue.

1. Si selezioni il controllo da spostare facendovi sopra clic. Per selezionare più controlli da spostare contemporaneamente, si faccia clic sugli stessi premendo e mantenendo premuto il tasto ⇧ (Maiusc).
2. Si faccia di nuovo clic sul controllo e, mantenendo premuto il pulsante del mouse, lo si trascini nella posizione desiderata.

Spostamento di menu

Per spostare un menu all'interno della Barra dei menu, si proceda come segue.

1. Si richiami la finestra Menu Editor facendo clic con il pulsante destro del mouse all'interno del form selezionando, dal menu rapido che viene aperto, il comando Menu Editor
2. Si selezioni il menu o il comando di menu che si desidera spostare facendo clic sul suo nome.
3. Si faccia clic sul pulsante con la freccia rivolta verso l'alto o verso il basso per spostare in alto o in basso la voce del menu o del comando selezionato. Nel corso dello spostamento l'elemento spostato manterrà il proprio livello originario di "rientro gerarchico".
4. Si ripetano i passi 2 e 3 per spostare le altre voci di menu o comandi.
5. Si faccia clic su OK per confermare gli spostamenti effettuati.

Nota. Quando si fa clic su OK della finestra di dialogo Menu Editor, Visual Basic controlla la correttezza della sequenza dei menu e dei corrispondenti comandi. Se Visual Basic rileva un errore, ne darà comunicazione indicando il livello nel quale questo si è verificato.

L'istruzione Print

L'istruzione Print visualizza un testo all'interno della finestra Immediata. (questa finestra consente di esaminare i valori delle variabili e verificare funzioni).

La sintassi generale di questa istruzione è la seguente.

```
oggetto.Print [outputlist]
```

La componente `oggetto` dell'enunciato indica l'oggetto che dovrà essere stampato (per esempio, `Debug`). L'argomento facoltativo `[outputlist]` corrisponde, invece, a un'espressione o a un elenco di espressioni da stampare. Quando si omette questa indicazione facoltativa, verrà visualizzata una riga vuota. L'argomento `outputlist` prevede la seguente sintassi

```
[Spc(n) ¦ Tab(n)] espressione posizionecarattere
```

La parte facoltativa `Spc(n)` inserisce una serie di caratteri "spazio" il cui numero viene definito da (n). L'elemento facoltativo `Tab(n)`, invece, inserisce un valore assoluto di colonne (definito da (n)) al quale dovrà essere spostato il cursore. Per spostare il cursore all'inizio della successiva area di stampa, si utilizzi `Tab` senza associarlo ad alcun argomento specifico. L'argomento facoltativo `espressione` può corrispondere a un'espressione sia numerica sia letterale.

L'argomento facoltativo `posizionecarattere` specifica il punto di inserimento immediatamente successivo l'ultimo carattere stampato. `Tab(n)` imposta il punto di inserimento a un valore numerico di colonna assoluto e se si omette l'argomento `posizionecarattere`, il carattere successivo verrà riprodotto all'inizio della riga seguente.

Viene ora proposto un esempio pratico dell'enunciato `Print`

```
Sub Stampalo(X As Double)
  Debug.Print "La radice quadrata di " ; X ;" = "; Sqrt(X)

End Sub
```

Questo esempio mostra la procedura `Stampalo` che visualizza il valore di tipo `Double` dell'argomento `X` e il valore della sua radice quadrata. La procedura, per visualizzare tali valori, utilizza l'enunciato `Debug.Print`. Viene anche visualizzata la stringa letterale `La radice quadrata di`, il valore dell'argomento `X`, la stringa letterale = e, infine, il valore della radice quadrata dell'argomento `X`. L'enunciato `Print` si serve del carattere punto e virgola (";") per concatenare gli elementi da visualizzare sulla stessa riga.

Selezione di uno o più controlli

Per selezionare un singolo controllo vi si faccia sopra clic con il mouse; per deselezionare un controllo precedentemente selezionato occorre fare clic sulla form.

Per selezionare contemporaneamente più controlli si proceda come segue.

1. Si selezioni il primo controllo della serie facendovi sopra clic.
2. Premere e mantenere premuto il tasto ⇧(Maiusc) e si faccia clic su tutti gli altri controlli da selezionare.

Funzioni per la gestione dell'ora

Visual Basic è dotato di una serie di funzioni che consentono di accedere all'orologio di sistema. Nei paragrafi di questa sezione si prenderà in considerazione ogni singola funzione di questo tipo.

La funzione Hour( )

La funzione `Hour( )` restituisce un `Variant(Integer)`, ovvero un valore intero compreso tra 0 e 23, che rappresenta la componente di `ora` corrispondente all'ora del giorno. L'argomento `ora` può essere un `Variant`, un'espressione numerica, letterale o una qualsiasi loro combinazione che possa rappresentare un'ora ammissibile.

La sintassi generale della funzione `Hour( )` è la seguente

Hour(ora)

Quello che segue è un esempio di un possibile utilizzo della funzione

```
Dim OrarioAttuale As Variant, OraAttuale As Integer
OrarioAttuale = #17:11:16# ' Assegna un'ora.
OraAttuale = Hour(OrarioAttuale)
```

Questo esempio, innanzi tutto, dichiara la variabile `OrarioAttuale` di tipo `Variant` e la variabile `OraAttuale` di tipo `Integer`. Nell'esempio, alla variabile `OrarioAttuale` viene assegnata una stringa che rappresenta un'ora (17:11:16, oppure 5:11:16PM in un sistema orario a 12 ore). Viene poi richiamata la funzione `Hour( )` in modo da poter estrarre l'ora dalla variabile `OrarioAttuale`. L'enunciato che richiama la funzione `Hour( )` memorizza quindi il risultato di questa funzione (17) all'interno della variabile `OraAttuale`.

La funzione Minute( )

La funzione `Minute( )` restituisce un `Variant(Integer)`, un valore intero compreso tra 0 e 59, che rappresenta la componente di `ora` corrispondente ai minuti. L'argomento `ora` può essere un `Variant`, un'espressione numerica, letterale o una qualsiasi loro combinazione che possa rappresentare un'ora ammissibile.

La sintassi generale della funzione `Minute( )` è la seguente

Minute(ora)

Quello che segue è un esempio di un possibile utilizzo della funzione `Minute( )`.

```
Dim OrarioAttuale As Variant, MinutoAttuale As Integer
OrarioAttuale = #17:11:16# ' Assegna un'ora.
MinutoAttuale = Minute(OrarioAttuale)
```

Questo esempio, innanzi tutto, dichiara la variabile `OrarioAttuale` di tipo `Variant` e la variabile `MinutoAttuale` di tipo `Integer`. Nell'esempio, alla variabile `OrarioAttuale` viene assegnata una stringa che rappresenta un'ora (17:11:16, oppure 5:11:16 PM in un sistema orario a 12 ore).

Viene poi richiamata la funzione `Minute( )` in modo da poter estrarre i minuti della variabile `OrarioAttuale`. L'enunciato che richiama la funzione `Minute( )` memorizza quindi il risultato di questa funzione (17) all'interno della variabile `MinutoAttuale`.

La funzione Second( )

La funzione `Second( )` restituisce un `Variant(Integer)`, un valore intero compreso tra 0 e 59, che rappresenta la componente di `time` corrispondente ai secondi del giorno. L'argomento `ora` può essere un `Variant`, un'espressione numerica, letterale o una qualsiasi loro combinazione che possa rappresentare un'ora ammissibile.

La sintassi generale della funzione `Second( )` è la seguente.

Second(ora)

Quello che segue è un esempio di un possibile utilizzo della funzione .

```
Dim OrarioAttuale As Variant, SecondoAttuale As Integer
OrarioAttuale = #17:11:16# ' Assegna un'ora.
SecondoAttuale = Second(OrarioAttuale)
```

Questo esempio, innanzi tutto, dichiara la variabile `OrarioAttuale` di tipo `Variant` e la variabile `SecondoAttuale` di tipo `Integer`. Nell'esempio, alla variabile `OrarioAttuale` viene assegnata una stringa che rappresenta un secondo (17:11:16, oppure 5:11:16 PM in un sistema orario a 12 ore). Viene poi richiamata la funzione `Second( )` in modo da poter estrarre i secondi della variabile `OrarioAttuale`. L'enunciato che richiama la funzione `Second( )` memorizza quindi il risultato di questa funzione (16) all'interno della variabile `SecondoAttuale`.

La funzione Time( )

La funzione `Time( )` restituisce un `Variant(Date)` che corrisponde all'ora attiva di sistema.

La sintassi generale della funzione `Time( )` è la seguente.

```
Time
```

Quello che segue è un esempio di un possibile uso della funzione.

```
Dim OrarioVariabile As Variant
OrarioVariabile = Time
```

In questo esempio viene dichiarata la variabile `OrarioVariabile` di tipo `Variant` nella quale si memorizza l'ora di sistema tramite il richiamo della funzione `Time( )`.

Uso della finestra "Immediata"

Visual Basic prevede la presenza della finestra Immediata per visualizzare e verificare i valori di un programma. Nel caso in cui questa finestra non sia visibile si scelga il comando Visualizza ⇨ Finestra Immediata. Gli scopi di tale finestra sono, principalmente, due.

- **Visualizzare i valori di un programma Visual Basic tramite l'istruzione `Debug.Print`.** Per esempio, se si desiderano verificare i valori di una variabile chiamata `Indice`, si potrà inserire in una procedura l'istruzione `Debug.Print Indice`. Questo enunciato, quando verrà eseguito, visualizzerà nella finestra Immediata il valore della variabile Indice.
- **Eseguire istruzioni scritte oppure incollate in un programma Visual Basic.** Per esempio, nella finestra Immediata si potrà scrivere direttamente `Print Indice` e premere Invio per visualizzare immediatamente il valore della variabile `Indice`.

Regole per l'assegnazione dei nomi

Visual Basic quando si desidera assegnare nomi a costanti, variabili, subroutine, funzioni, tipi di dati personalizzati, classi eccetera, richiede che vengano adottate alcune regole precise.

- Il **primo carattere** di un nome Visual Basic deve necessariamente essere una lettera.

- I **caratteri successivi** potranno essere lettere, numeri o caratteri "_" (*underscore*).
- I nomi Visual Basic **non fanno distinzione fra maiuscole e minuscole**.

Qui di seguito vengono forniti alcuni esempi di nomi ammissibili.

```
X
HARI_KARI
FiltroAmore9
mioNome_01
```

Ricordare. Dato che Visual Basic non rileva la differenza fra l'uso di caratteri maiuscoli o minuscoli, i nomi `FiltroAmore9`, `FILTRO AMORE9` e `fILtroaMORe9` verranno sempre riconosciuti come *lo stesso nome*.

Per maggiori informazioni circa le regole di assegnazione dei nomi, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Integrazione con Visual Studio

Visual Basic 6 fa parte del pacchetto di sviluppo Visual Studio, che include, fra l'altro, anche Visual C++ e Visual J++. Nonostante l'interfaccia utente di Visual Basic 6 sia differente da quella di Visual C++ e Visual J++, questi prodotti sono integrati nei seguenti aspetti.

- È possibile sviluppare controlli ActiveX in Visual Basic e utilizzarli in progetti Visual C++ e Visual J++. È inoltre possibile utilizzare in progetti Visual Basic i controlli ActiveX realizzati in Visual C++ e Visual J++.
- È possibile utilizzare in progetti Visual C++ e Visual J++ finestre di dialogo realizzate in Visual Basic. Tuttavia tale possibilità è sconsigliata a causa del fatto che i controlli Visual Basic possono essere gestiti da Visual C++ e Visual J++ in maniera molto limitata.
- Si può utilizzare Visual J++ per creare applicazioni Windows con lo stesso approccio di Visual Basic. Ciò significa che anche Visual J++ consente di disegnare controlli sulle form e di associarvi delle funzioni. Visual C++, al contrario, salva le proprie maschere, finestre di dialogo e controlli in un apposito file di risorse.

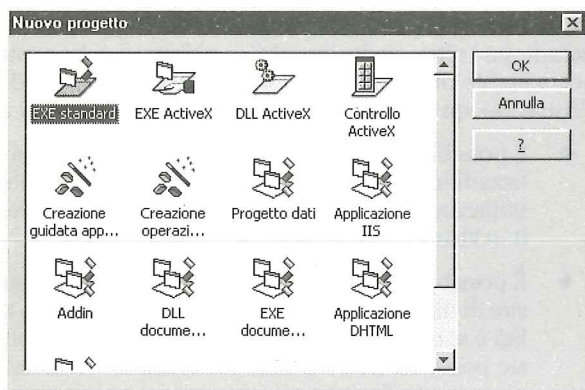
Il fatto che Visual J++ abbia adottato tecniche di programmazione visuale simili a quelle presenti in Visual Basic segnala la tendenza di Visual Studio ad avvicinarsi alla filosofia di tale ambiente di svi-

luppo. Ci si può quindi aspettare anche da parte di Visual C++, per la prossima versione di Visual Studio, un maggiore supporto allo stile di sviluppo presente in Visual Basic. Inoltre, ci si augura che le prossime versioni di ogni ambiente di sviluppo aumentino la compatibilità dei controlli.

I file di progetto

I menu di Visual Basic contengono diversi comandi che consentono al programmatore di interagire con i vari file di progetto.

Creazione di un nuovo progetto

Il comando File ⇨ Nuovo progetto consente di creare un nuovo file di progetto al quale viene automaticamente assegnata l'estensione ".VBP". Questo comando richiama la finestra di dialogo Nuovo progetto che consente di selezionare il tipo di progetto da avviare. La figura seguente riporta un esempio di questa finestra per la versione Enterprise di Visual Basic. Come si potrà notare si ha la possibilità di avviare nove tipi di progetto ma, nella maggior parte dei casi, si sceglierà l'opzione `Standard EXE`.

Aggiunta di un progetto

Il comando File ⇨ Aggiungi progetto permette di aggiungere al file di progetto .VBP più di un progetto. Si utilizzi questo comando per combinare, in un solo file .VBP, più progetti. Il comando Aggiungi progetto genera l'apertura della finestra di dialogo Aggiungi progetto.

Rimozione di un progetto

Per eliminare un progetto lo si selezioni facendo clic sul suo nome nella finestra Progetto e avviando quindi il comando File ⇨ Rimuovi progetto. Visual Basic rimuove il progetto e questa modifica viene riflessa anche nella finestra Progetto.

Salvataggio di un progetto

Per salvare un progetto si hanno ha disposizione due alternative. Il comando File ⇨ Salva progetto salva il progetto e i file collegati assegnandogli il nome corrente. Quando si usa questo comando su un nuovo progetto, Visual Basic richiede se si desidera assegnare al file di progetto un nuovo nome.

Il comando File ⇨ Salva progetto con nome salva il progetto sul quale si sta lavorando consentendo di assegnargli un nuovo nome.

Aggiunta al programma di form e di altri elementi

Il menu Progetto contiene i seguenti comandi.

- Inserisci form permette di inserire nel progetto un nuovo form.
- Inserisci form MDI *consente di aggiungere al progetto un form MDI (Multiple Document Interface).*
- Inserisci modulo inserisce in un progetto un nuovo modulo.
- Inserisci modulo di classe permette di inserire un modulo classe.
- Inserisci controllo utente consente di aggiungere controlli di tipo personalizzato.
- Inserisci pagina proprietà inserisce una pagina per le proprietà.
- Inserisci documento utente permette di inserire documenti di tipo personalizzato.
- Inserisci DHTML page permette di inserire una pagina Dynamic HTML.
- Inserisci Data Report permette di inserire un report.
- Inserisci WebClass consente di inserire una classe orientata al Web.
- Inserisci Microsoft User Connection consente di aggiungere una connessione a database.

- Inserisci altre finestre di progettazione ActiveX permette di aggiungere oggetti ActiveX.
- Inserisci file consente di inserire file esterni.

Rimozione di un file

Per rimuovere da un progetto un form, un modulo di classe o altri elementi si selezioni, dalla finestra Progetto la voce corrispondente e si scelga il comando Progetto (associato al nome dell'elemento da rimuovere). Per esempio, se si è selezionato il form `Form2`, il menu Progetto dovrebbe contenere il comando Rimuovi Form2.

La finestra Progetto

La finestra di progetto (chiamata anche *Gestione progetti*) contiene un elenco gerarchico di tutti i progetti e degli elementi nello stesso contenuti, come esemplificato nella figura di pagina seguente.

La finestra di progetto è composta dai seguenti elementi.

- Il pulsante Visualizza codice apre una finestra che visualizza il codice sorgente del progetto nel quale si potrà intervenire per apportare le opportune modifiche o aggiustamenti.
- Il pulsante Visualizza oggetto apre la finestra relativa alla voce selezionata, a un form esistente, a un oggetto ActiveX o a un controllo utente.
- Il pulsante Espandi/comprimi cartelle consente di visualizzare la cartella contenente i veri elementi di un progetto.
- La finestra del progetto mostra tutti i progetti caricati e gli elementi che ne fanno parte.

In particolare, quest'area contiene le voci seguenti.

- **Progetto:** il progetto e gli elementi che lo compongono.
- **Form:** tutti i form (file con estensione ".FRM") associati al progetto.

- **Moduli:** nomi dei moduli (file con estensione ".BAS") utilizzati nel progetto.
- **Moduli di classe:** nomi di moduli di classe (file con estensione ".CLS") utilizzati nel progetto.
- **Controlli utente:** tutti i controlli utilizzati nel progetto.
- **Documenti utente:** (file con estensione ".DOB") che fanno parte del progetto.
- **Pagine delle proprietà:** (file con estensione ".PAG") nel progetto.
- **Documenti correlati:** documenti ai quali si vuole accedere dall'interno del progetto.
- **Risorse:** tutte le risorse disponibili nel progetto.

La finestra proprietà

Si faccia riferimento alla sezione di questa parte che descrive come personalizzare le proprietà.

Controlli semplici

In questa parte vengono esaminati i controlli di tipo più semplice. Visual Basic mette a disposizione controlli per eseguire, virtualmente, qualsiasi operazione: dai semplici pulsanti o ai messaggi (presi in esame in questa parte) ai controlli di tipo più avanzato descritti nella *Parte V*.

Utilizzando queste due parti si dovrebbe essere in grado di controllare qualsiasi elemento di un'applicazione.

Argomenti trattati

- ✔ **Uso dei controlli per consentire agli utenti di inserire i propri dati**
- ✔ **Uso dei controlli per creare pulsanti, caselle di controllo e altri elementi necessari**
- ✔ **Controlli per la visualizzazione del testo e dei messaggi**

Controllo "Pulsante Bitmap"

Un controllo Pulsante Bitmap (Bitmap Button) permette al programmatore di inserire immagini bitmap (o icone) all'interno dei pulsanti di un'applicazione Visual Basic.

Nella tabella seguente vengono descritte schematicamente le proprietà rilevanti del controllo corrispondente ai pulsanti di comando che consentono di utilizzare le immagini bitmap come veri e propri pulsanti.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commenti*
Style: `oggetto.`**`Style`**	Restituisce o imposta i valori che indicano il tipo di visualizzazione e il comportamento del controllo. In fase di run-time si tratta di una proprietà di sola lettura. Il valore `vbButton Graphical(1)` trasforma il controllo pulsante in un pulsante bitmap; al contrario, il valore `vbButton Standard(0)` applica al controllo lo stile standard.	`btnStyle = cmdQuit.Style`	Memorizza la proprietà `Style` del pulsante `cmdQuit` all'interno della variabile `btn-Style`.
Picture: `oggetto.`**`Picture`** `[= immagine]`	Restituisce o imposta una immagine che dovrà essere visualizzata all'interno del controllo. Per default un pulsante bitmap non contiene alcuna immagine.	`cmdQuit.Picture = LoadPicture ('quit.bmp')`	Imposta la proprietà `Picture` caricando il file bitmap QUIT .BMP.
DownPicture: `oggetto.`**`DownPicture`** `[= immagine]`	Restituisce o imposta i riferimenti a un'immagine che deve essere visualizzata in un controllo pulsante quando l'utente fa clic e mantiene premuto il pulsante del mouse.	`cmdQuit.DownPicture = LoadPicture ('quitdn.bmp')`	Imposta la proprietà `DownPicture` caricando il file bitmap QUITDN.BMP.
DisablePicture: `oggetto.`**`DisablePicture`** `[= immagine]`	Restituisce o imposta i riferimenti a un'immagine da visualizzare in un controllo quando questo è disabilitato.	`cmdQuit.Disable Picture = LoadPicture ('quiddp.bmp')`	Imposta la proprietà `Disable Picture` caricando il file bit map `QUITDP.BMP`.

Per maggiori informazioni sul pulsante bitmap, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Viene ora presentato un esempio di programma che utilizza il pulsante bitmap esemplificato nella figura.

In questo esempio vengono visualizzati il pulsante bitmap "Premi!" e il pulsante di comando "Disabilita". Quando l'utente farà clic sul pulsante bitmap, il programma visualizzerà un semplice messaggio testuale; al contrario, se l'utente farà clic e manterrà premuto il pulsante del mouse sul pulsante bitmap, nel pulsante stesso verrà visualizzata l'immagine associata alla proprietà `DownPicture`. Quando l'utente farà clic sul pulsante "Disabilita", il pulsante bitmap verrà disabilitato e verrà visualizzata l'immagine bitmap associata alla proprietà `DisablePicture`. Quando l'utente farà clic sul pulsante "Abilita", il pulsante verrà di nuovo abilitato e visualizzerà l'immagine bitmap associata alla proprietà Picture.

La tabella seguente mostra le impostazioni del form e dei controlli dell'esempio.

Modulo/Controllo	*Proprietà*	*Impostazione*
Form	Name	Form1 (default)
	Caption	Pulsante Bitmap
Pulsante di comando	Name	cmdClickMe
	Caption	Premi!
	Picture	UPBTN.BMP
	DownPicture	DNBTN.BMP
	DisablePicture	DSBTN.BMP
Pulsante di comando	Name	cmdDisable
	Caption	&Disabilita

Ecco il codice sorgente.

```
Private Sub cmdClickMe_Click()
  MsgBox "Ciao!", , "Saluti"
End Sub
Private Sub cmdDisable_Click()
  If cmdClickMe.Enabled Then
   cmdClickMe.Enabled = False
   cmdDisable.Caption = "&Abilita"
  Else
   cmdClickMe.Enabled = True
   cmdDisable.Caption = "&Disabilita"
  End If
End Sub
```

La procedura `cmdClickMe_Click( )` risponde a un'operazione di clic sul pulsante bitmap visualizzando il messaggio `Ciao!`. La procedura `cmdDisable_Click( )` risponde a un clic sul pulsante di comando "Disabilita". Questo gestore dell'evento esamina la proprietà `Enabled` del pulsante di comando per determinare se abilitare o disabilitare il pulsante bitmap. Quando la proprietà `Enabled` del pulsante di comando è `&Disabilita` ha valore vero, il gestore di evento esegue le operazioni seguenti.

- Disabilita il pulsante bitmap.
- Imposta la proprietà `Caption` del pulsante di comando su `&Abilita`.

Se la proprietà `Enabled` del pulsante ha valore falso, il gestore dell'evento esegue le operazioni seguenti.

- Abilita il pulsante bitmap.
- Imposta la proprietà `Caption` su `&Disabilita`.

Controllo "Pulsante di comando"

Un Pulsante di comando (Comman Button) è, probabilmente, uno dei controlli più usati nelle applicazioni Windows. L'utente, di solito, fa clic su un pulsante di comando per eseguire un'operazione come, per esempio, quella per la lettura di un testo da un file, calcolare un valore eccetera.

Nella tabella seguente vengono riportate le proprietà rilevanti di un controllo Pulsante di comando quando viene utilizzato come pulsante di tipo standard. Gli eventi rilevanti per il pulsante di comando sono `Click`, `LostFocus` e `GotFocus`.

Proprietà: sintassi	Scopo	Esempio	Commenti
Caption: `oggetto.Caption [= stringa]`	Restituisce o imposta l'etichetta del pulsante di comando.	`cmdDisable.Caption = "&Abilita"`	Imposta a `&Abilita` l'etichetta del pulsante `cmdDisable`.
Enabled: `oggetto.Enabled [= booleano]`	Restituisce o imposta lo stato di abilitazione del pulsante di comando.	`cmdExit.Enabled = Not Enabled`	Imposta la proprietà `Enabled` del pulsante di comando `cmd Exit`.
Visible: `oggetto.Visible [= booleano]`	Restituisce o imposta lo stato di visibilità del pulsante di comando.	`cmdExit.Visible = Not cmdExit.Visible`	Imposta la proprietà `Visible`del pulsante di comando `cmd Exit`.
Default: `oggetto.Default [= booleano]`	Restituisce o imposta lo stato di default del pulsante di comando. Il modulo richiama il valore di default ogni volta che si preme Invio.	`cmdExit.Default = Not cmdExit. Default`	Imposta la proprietà `Default` del pulsante di comando `cmd Exit`

Per un esempio di come usare un pulsante di comando, si veda l'esempio riportato nella sezione di questa parte che descrive il controllo "Pulsante Bitmap".

Controllo "Casella di controllo"

I controlli Casella di controllo (Check Box) sono composti da un piccolo quadrato che riporta una `x` quando l'utente desidera selezionare l'opzione associata (definita da una didascalia che ne descrive l'operatività). Quando si disattiva la casella di controllo, la `x` scompare. Ci si serve di questo controllo per consentire all'utente di attivare o disattivare alternativamente ("vero/falso", oppure "si/no") una determinata opzione. Le caselle di controllo potranno essere inserite in gruppi di opzioni omogenee che consentano di selezionare, all'interno dello stesso gruppo, più opzioni contemporaneamente. Le caselle di controllo non si escludono l'una con l'altra, ovvero l'attivazione o disattivazione di un'opzione di un gruppo non influisce sulle altre opzioni dello stesso tipo e dello stesso gruppo.

Le proprietà rilevanti delle caselle di controllo sono `Value` e `Caption`.

Proprietà Value

La sintassi generale per l'uso della proprietà `Value` di una casella di controllo è la seguente.

```
checkBoxControl.Value [= valore]
```

La proprietà `Value` indica lo stato di una casella di controllo e i valori ammissibili sono `0` (casella vuota: opzione deselezionata), `1` (casella e opzione corrispondente selezionate) e `2` (opzione temporaneamente non disponibile). La proprietà `Value` può essere usata per impostare o per verificare lo stato attivo di una casella di controllo.

Quello che segue è un esempio di come modificare lo stato di una casella di controllo.

```
If chkBackup.Value = 0 Then
  chkBackup.Value = 1
Else
  chkBackup.Value = 0
End If
```

Questo esempio utilizza un enunciato `If` per determinare lo stato di disattivazione della casella di controllo (casella vuota). In questo caso (`Value = 0`) il codice assegna alla proprietà `Value` del controllo il valore `1` (ovvero attiva la casella). In caso contrario, l'enunciato `If` assegna alla proprietà `Value` il valore `0` (ovvero disattiva [svuota] la casella di controllo). Si noti che non viene utilizzato l'operatore `NOT` per modificare lo stato della casella di controllo. Infatti, la casella di controllo ha un terzo stato possibile: disabilitato.

Proprietà Caption

La proprietà `Caption` visualizza, sotto forma di stringa, il testo descrittivo dell'opzione associata alla casella di controllo.

La sintassi generale della proprietà `Caption` è la seguente.

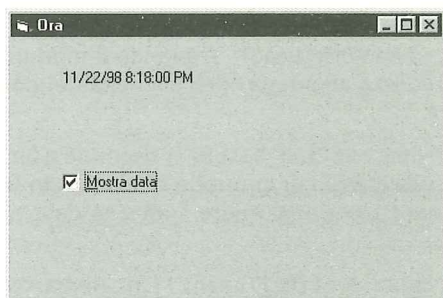
```
checkBoxControl.Caption [= stringa]
```

Quello che segue è un esempio di come la proprietà `Caption` possa disabilitare la casella di controllo associata all'opzione "Salvataggio automatico".

```
Dim CasellaControllo, SeriePersonale
For Each CasellaControllo In SeriePersonale
  If CasellaControllo.Caption = "Salvataggio automatico"
Then
    CasellaControllo.Value = 0 ' disattiva caselle
```

```
    Exit For          ' Esce dal ciclo
  End If
Next CasellaControllo
```

Questo esempio utilizza il ciclo `For Each` per ricercare la casella di controllo la cui proprietà `Caption` è "Salvataggio automatico". Quando il ciclo la trova, imposta la corrispondente proprietà `Value` a `0`.

Si osservi ora l'esempio della figura seguente nella quale appare una casella di controllo per la visualizzazione dell'ora e della data di sistema correnti. Inizialmente, il modulo visualizza l'ora corrente, ma quando l'utente seleziona la casella di controllo verranno visualizzate sia la data sia l'ora. Al contrario, quando l'utente deselezionerà l'opzione, il modulo visualizzerà solo l'ora corrente.

La tabella seguente riepiloga i dati relativi al modulo e ai controlli e le corrispondenti impostazioni delle proprietà.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Ora
Etichetta	Name	lblTime
Temporizzatore	Name Interval	tmrTime 1000
Casella controllo	Name Caption	chkTime &MostraData

Quello che segue è il codice sorgente del programma di esempio.

```
Option Explicit
Private Sub ShowTime()
  Dim TStr As Variant
  TStr = Time
  If chkTime.Value = 1 Then
```

```
    TStr = TStr + Date
  End If
  lblTime.Caption = TStr
End Sub
Private Sub chkTime_Click()
  ShowTime
End Sub
Private Sub Form_Load()
  lblTime.Caption = Time
End Sub
Private Sub tmrTime_Timer()
  ShowTime
End Sub
```

Osservando il contenuto del codice si noterà la presenza dei seguenti gestori di evento.

- La procedura `Form_Load( )`, quando il modulo viene caricato, visualizza all'interno dell'etichetta del controllo, l'ora corrente.
- La procedura `chkTime_Click( )` risponde a un'operazione di clic sulla casella di controllo richiamando la procedura `ShowTime( )` per aggiornare l'ora e la data nel controllo etichetta.
- La procedura `tmrTime_Timer( )` richiama, ogni secondo che passa, la procedura `ShowTime( )` e aggiorna l'ora e la data all'interno del controllo etichetta.
- La procedura `ShowTime( )` assegna l'ora corrente alla proprietà `Caption` del controllo etichetta solo nel caso in cui la casella di controllo sia vuota. Al contrario, se la casella è selezionata, la procedura inserisce la data corrente nella proprietà `Caption` del controllo etichetta.

Controllo "Casella combinata"

Il controllo Casella combinata (Combo box) è un controllo composto da una casella di elenco di voci e da un'area di immissione. Questo tipo di controllo consente all'utente sia di selezionare una voce predefinita, sia di inserirne una manualmente.

Nella tabella seguente vengono schematizzate le proprietà rilevanti del controllo Casella combinata.

Proprietà: sintassi	*Oggetto*	*Proprietà*	*Impostazione*
List: i.cboBox.**List**(indice) [= stringa]	Restituisce o imposta le voci predefinite contenute nell'elenco del controllo.	cboNames.List(0) = "Guglielmo"	Assegna il nome `Guglielmo` alla prima voce dell'elenco della casella combinata.
ListCount: cboBox.**ListCount**	Restituisce il numero delle voci contenute nell'elenco della casella combinata.	`If cboNames. ListCount > 0 Then CboNames. List(0) = "Guglielmo"`	Nel caso in cui la parte l'elenco della casella combinata non sia vuoto, sovrascrive la prima voce dell'elenco.
ListIndex: cboBox.**ListIndex** [= indice]	Recupera o imposta l'indice della voce selezionata all'interno del controllo. Non è disponibile in fase di progettazione del controllo. Quando non viene effettuata una selezione, la proprietà assume il valore –1.	`SelIdx = cboNames. ListIndex`	Memorizza nella variabile `SelIdx` l'indice della voce selezionata.
Sorted: cboBox.**Sorted**	Determina se le voci nella casella combinata sono state ordinate.	`If cboNames. Sorted Then aggiungi_ MiaVoce "io" Else cboNames. AddItem "io"`	Esamina la proprietà `Sorted` per determinare se usare una procedura personalizzata per aggiungere "io" oppure avvia il metodo `AddItem( )` per aggiungerla.
Style: cboBox.**Style** ***Nota.*** Si tratta di una proprietà di sola lettura.	Determina lo stile della casella combinata.Il valore 0(`vbComboDropDown`) corrisponde all'impostazione di default e fa sì che il controllo appaia come un elenco a comparsa. Il valore 1 (`vbComboSimple`) fa si che il controllo appaia come una normale casella combinata con visualizzazione dell'elenco delle voci. Il valore 2 (`vbComboDropDownList`) genera una casella combinata con un elenco di voci a scorrimento.	`For Each Ctrl In mioGruppoCbo If Ctrl. style = 0 Then Ctrl AddItem "io", 1 End If Next Ctrl`	Aggiunge a ogni casella combinata che ha lo stile di default, la stringa "io".

continua

Proprietà: sintassi	*Oggetto*	*Proprietà*	*Impostazione*
Text: `cboBox.`**`Text`** `[= testo]`	Restituisce o imposta il testo contenuto nell'area di modifica (quando la proprietà Style è impostata a `1` o `0`). Restituisce la voce selezionata nell'elenco se la proprietà Style è impostata a `2`.	`cboNames.Text = io`	Assegna la stringa "`io`" all'area di immissione della casella combinata (presupponendo che la proprietà Style non sia impostata a 2).
SelLenght: `cboBox.`**`SelLength`** `[= numero]`	Restituisce o imposta il numero dei caratteri selezionati all'interno dell'area di immissione della casella combinata.	`cboNames.Sel Length = 0`	Disattiva la selezione di caratteri effettuata nell'area di immissione della casella combinata.
SelStart: `cboBox.`**`SelStart`** `[= indice]`	Restituisce o imposta l'indice del primo carattere selezionato nell'area di immissione della casella combinata.	`cboNames.Sel Start = 1`	Fa sì che la selezione dell'area di immissione della casella combinata inizi dal primo carattere.
SelText: `cboBox.`**`SelText`** `[= testo]`	Restituisce o imposta il testo selezionato.	`If cboNames. SelText = "" Then cboNames.Sel Start = 1 cboNames.Sel Length = 1 End If`	Seleziona il primo carattere dell'area di immissione nel caso non ne sia stato selezionato nessun altro.

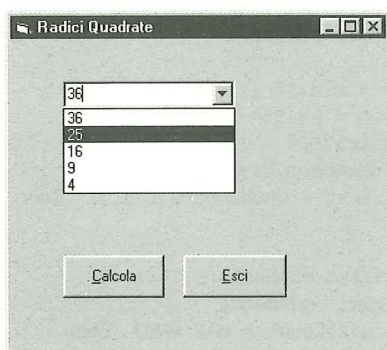
Nella tabella seguente vengono schematicamente descritti i metodi associati al controllo Casella combinata.

Metodo: sintassi	*Descrizione*	*Esempio*	*Commento*
AddItem: `cboBox.`**`AddItem`** `voce, indice`	Aggiunge una nuova voce all'elenco della casella combinata.	`cboIo.AddItem "io", 0`	Aggiunge la stringa "`io`" come prima voce della componente elenco della casella combinata.
Clear: `cboBox.`**`Clear`**	Elimina le voci dalla componente elenco della casella combinata.	`cboNames. Clear`	Elimina le voci dalla componente elenco della casella `cboNames`.
RemoveItem: `cboBox.`**`RemoveItem`** `indice`	Rimuove la voce specificata da indice dalla componente elenco della casella combinata.	`cboIo.Remove Item 0`	Rimuove la prima voce dalla componente elenco della casella combinata `cboIo`.

Infine, nella tabella seguente si indicano gli eventi rilevanti associabili a un controllo Casella combinata.

Evento	*Descrizione*
`Click`	Si verifica quando si fa clic all'interno della componente elenco della casella combinata.
`DblClick`	Si verifica quando si fa doppio clic all'interno della componente elenco della casella combinata. Viene di solito usato per eseguire un'operazione su un elemento dell'elenco.
`Change`	Si verifica quando si modifica il contenuto di una casella combinata (la modifica si riflette sia nell'area di immissione sia in quella dell'elenco delle voci).
`Scroll`	Si verifica quando si fa scorrere il contenuto della componente elenco della Casella combinata.

Per maggiori informazioni sul controllo "Casella combinata", si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1988).

La figura è un esempio che fa uso di un controllo Casella combinata per la memorizzazione di un elenco dinamico di numeri.

Il programma utilizza una Casella combinata per immettere e memorizzare valori numerici. Il pulsante di comando `Calcola` preleva il valore numerico presente nell'area di immissione e ne calcola la radice quadrata, visualizzando quindi il numero e il risultato dell'operazione nell'etichetta posta sotto la casella combinata.

Nella tabella seguente viene schematizzata la struttura del form e dei controlli, indicandone le impostazioni.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Radice quadrata
Casella combinata	Name	cboNumbers
Etichetta	Name	lblResult
Pulsante di comando	Name Caption	cmdCalculate &Calcola
Pulsante di comando	Name Caption	cmdQuit &Esci

Quello che segue è il listato del codice sorgente per la generazione della figura precedente.

```
Option Explicit
Const MAX_NUMS = 5
Private Sub cmdCalc_Click( )
  Dim I As Integer
  Dim X As Double
  Dim ResStr As String

  If cboNumbers.Text < > "" Then
    ' preleva il numero dall'area di immissione
    X = Abs(Val(cboNumbers.Text))
    ResStr = "Sqrt(" + Mid(Str8X), 2) + ") = "
    ' calcola la radice quadrata
    X = Sqr(X)
    resStr = ResStr + Mid(Str(X), 2)
    lblResult.Caption = ResStr
    ' ricerca i duplicati
    For i = 0 To cboNumbers.ListCount -1
      If cboNumbers.Text = cboNumbers.List(I) Then
        Exit Sub
      Next I
      cboNumbers.AddItem cboNumbers.Text, 0
      ' rimuove 'vecchi' elementi
      If cboNumbers.ListCount > MAX_NUMS Then
       cboNumbers.Remove.Item cboNumbers.ListCount -1
    End If
  End If
End Sub
Private Sub Form_Load( )
  cboNumbers.Text = ""
  lblResult.Caption = ""
End Sub
Private Sub cmdQuit_Click( )
  End
End Sub
```

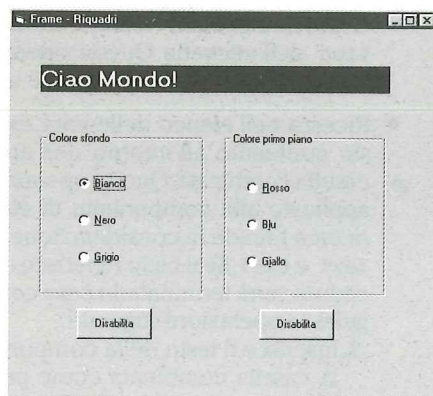
Questo codice sorgente dichiara la costante `MAX_NUMS` che specifica il numero massimo di voci da memorizzare nella componente elenco della casella combinata. Questo codice, inoltre, prevede la presenza dei seguenti gestori di evento.

- La procedura `Form_Load( )` analizza l'area di immissione della casella combinata e assegna una stringa vuota (`""`) alla proprietà `Caption` dell'etichetta.
- La procedura `cmdQuit_Click( )` risponde alla pressione del pulsante di comando `Esci`.
- La procedura `cmdCalculate_Timer( )` risponde alla pressione del pulsante di comando `Calcola` ed esegue i seguenti passi.
  - Preleva il testo contenuto nell'area di immissione della casella combinata e lo converte in un numero di tipo `Double`. Questa operazione è possibile accedendo al testo con la proprietà `Text` della casella combinata memorizzando quindi il numero nella variabile locale `X`.
  - Costruisce la prima parte del testo dell'etichetta inserendola nella variabile `StrRes`.
  - Calcola la radice quadrata del numero contenuto nella variabile `X` e memorizza il risultato di nuovo nella variabile `X`.
  - Genera la seconda parte del testo dell'etichetta inserendola nella variabile stringa `StrRes`.
  - Copia il testo della variabile `StrRes` nella proprietà `Caption` dell'etichetta. Questa operazione visualizza il risultato dell'estrazione della radice quadrata.
  - Ricerca nell'elenco delle voci, eventuali duplicati del testo contenuto all'interno dell'area di immissione della casella combinata. Questa operazione utilizza un ciclo `For` applicato alla componente di elenco del controllo. Tale ricerca prende in considerazione le proprietà `ListCount`, `Text` e `List`. Se il ciclo reperisce una concordanza, la procedura verrà terminata. In caso contrario, la procedura eseguirà le operazioni seguenti.
    1. Inserisce il testo della componente di immissione della casella combinata come prima voce della componente elenco di voci. Questa operazione fa uso del metodo `AddItem( )` associandolo ai valori di argomento `cboNumbers. Text` e `0`.
    2. Determina se la nuova voce numerica è superiore al valore definito dalla costante `MAX_NUMS`. Quando tale condizione è vera, la procedura rimuove l'ultima voce

della componente elenco della casella combinata servendosi del metodo `RemoveItem( )` e imposta il valore di argomento `cboNumbers.ListCount -1`

Controllo "Riquadro"

Un controllo Riquadro (Frame) viene utilizzato per raggruppare elementi quali, per esempio, caselle di controllo o pulsanti di opzione. Un controllo Riquadro può anche essere utilizzato per suddividere un form in funzioni operative distinte. Quando si crea un controllo in un Riquadro, il Riquadro verrà considerato come elemento contenitore e "proprietario" del controllo in esso inserito. Pertanto, se il controllo Riquadro viene disabilitato o nascosto, questa operazione influirà anche su tutti i controlli nello stesso contenuti.

Se si disabilita un controllo "Riquadro" si disabilitano anche i singoli controlli dipendenti anche se questi, in effetti, non appariranno in grigio. Per intervenire sull'aspetto dei controlli dipendenti da una struttura sarà necessario, pertanto, operare singolarmente sugli stessi servendosi di codice che eviti all'utente possibili fraintendimenti. Si osservi la figura seguente nella quale sono contenuti alcuni controlli Riquadro per la gestione dei colori di sfondo o di primo piano del testo contenuto in un'etichetta. Il form qui esemplificato contiene l'etichetta "Ciao Mondo!".

Nell'esempio il form contiene due oggetti Riquadro distinti: il primo controlla il colore di sfondo dell'etichetta tramite i pulsanti di opzione `Bianco`, `Nero` e `Grigio` mentre il secondo gestisce il colore del testo, servendosi, in questo caso, dei pulsanti di opzione `Ros-`

so, Blu e Giallo. Ogni controllo "Riquadro" prevede un proprio pulsante di comando (Disabilita) posto sotto la struttura stessa.

Quando l'utente farà clic su uno qualsiasi dei due pulsanti Disabilita, la struttura associata al pulsante stesso verrà completamente disabilitata e l'etichetta del pulsante verrà automaticamente modificata in Abilita. Quando l'utente farà clic su uno dei pulsanti di comando Abilita, la struttura e i controlli corrispondenti verranno di nuovo abilitati e il pulsante stesso riporterà la dicitura Disabilita.

Nella tabella seguente vengono schematizzati gli elementi del form, i relativi controlli e le loro impostazioni.

Oggetto	*Proprietà*	*Impostazione*
Form	Name	Form1 (default)
	Caption	Frame-Riquadri
Etichetta	Name	lblText
Frame	Name	grpBColor
	Caption	Colore sfondo
Pulsante opzione	Name	optWhite
	Caption	&Bianco
Pulsante opzione	Name	optBlack
	Caption	&Nero
Pulsante opzione	Name	optGray
	Caption	&Grigio
Pulsante comando	Name	cmdDisableBC
	Caption	Disabilita
Frame	Name	grpFColore
	Caption	Colore primo piano
Pulsante opzione	Name	optRed
	Caption	&Rosso
Pulsante opzione	Name	optYellow
	Caption	&Giallo
Pulsante opzione	Name	optBlue
	Caption	&Blu
Pulsante comando	Name	cmdDisableFC
	Caption	Disabilita

Il codice sorgente per la realizzazione di questo programma è il seguente.

```
Private Sub cmdDisableBC_Click( )
  If cmdDisableBC.Caption = "Abilita" Then
    grpBColor.Enabled = False
    cmdDisableBC.Caption = "Disabilita"
  Else
    grpBColor.Enabled = True
    cmdDisableBC.Caption = "Disabilita"
  End If
  ' abilita/disabilita i pulsanti di opzione
  ' nella medesima struttura contenitiva
  optWhite.Enabled = grpBColor.Enabled
  optBlack.Enabled = grpBColor.Enabled
  optGray.Enabled = grpBColor.Enabled
End Sub
Private Sub cmdDisableFC_Click( )
  If cmdDisableFC.Caption = "Disabilita" Then
    grpFColor.Enabled = False
    cmdDisableFC.Caption = "Abilita"
  Else
    grpFColor.Enabled = True
    cmdDisableFC.Caption = "Disabilita"
  End If
  ' abilita/disabilita i pulsanti di opzione
  ' della struttura contenitiva corrispondente
  optRed.Enabled = grpFColor.Enabled
  optBlue.Enabled = grpFColor.Enabled
  optYellow.Enabled = grpFColor.Enabled
End Sub
Private Sub Form_Load( )
  ' imposta l'etichetta e la dimensione del carattere
  lblText.Caption = "Ciao Mondo!"
  lblText.Font.Size = 24
  ' seleziona i pulsanti di opzione
  optBlack.Value = True
  optYellow.Value = True
End Sub
Private Sub optBlack_Click( )
  lblText.BackColor = vbBlack
End Sub
Private Sub optBlue_Click( )
  lblText.ForeColor = vbBlue
End Sub
Private Sub optGray_Click( )
  lblText.BackColor = vbGray
End Sub
Private Sub optRed_Click( )
  lblText.ForeColor = vbRed
```

```
End Sub
Private Sub optWhite_Click( )
  lblText.BackColor = vbWhite
End Sub
Private Sub optYellow_Click( )
  lblText.ForeColor = vbYellow
End Sub
```

La procedura `Form_Load( )` inizializza il modulo eseguendo le seguenti operazioni.

- Imposta la proprietà `Caption` dell'etichetta su `"Ciao Mondo!"`.
- Imposta la dimensione del carattere (*font*) del testo dell'etichetta a 24 punti.
- Seleziona il pulsante di opzione Nero del controllo Riquadro" Colore sfondo.
- Seleziona l'opzione Red del controllo"Riquadro"Colore primo piano.

Le procedure `cmdWhite_Click( )`,`cmdBlack_Click( )` e `cmdGray_Click( )` rispondono alle operazioni di clic dell'utente sui corrispondenti pulsanti di opzione della struttura `Colore sfondo` assegnando i colori relativi alla proprietà `BackColor` del controllo etichetta.

Le procedure `cmdRed_Click( )`,`cmdBlue_Click( )` e `cmdYellow_Click( )` rispondono alle operazioni di clic dell'utente sui corrispondenti pulsanti di opzione e assegnano i colori relativi alla proprietà `ForeColor` del controllo etichetta.

Le procedure `cmdDisableBC_Click( )` e `cmdDisableFC_Click( )` rispondono alle operazioni di clic da parte dell'utente sui pulsanti comando Disabilita posti sotto i due controlli Riquadro avviando le seguenti operazioni.

- Esamina la proprietà `Caption` del pulsante di comando per determinare lo stato di abilitazione dell'oggetto Riquadro associato.Questa operazione si serve di un enunciato `If` che modifica la proprietà `Enabled` della struttura e aggiorna, di conseguenza, la proprietà `Caption` del pulsante di comando.
- Copia il nuovo valore della proprietà `Enabled` dell'oggetto Riquadro nelle proprietà `Enabled` dei pulsanti di opzione contenuti nella struttura.Questa operazione consente di fare apparire in grigio (ovvero temporaneamente non disponibili) le opzioni tutte le volte che la struttura che li contiene viene disabilitata.

Controllo "Elenco di voci"

Il controllo Elenco di voci (List Box) visualizza un elenco di voci dal quale l'utente potrà selezionarne una o più di una, a seconda dello stile definito per l'interfaccia utente. L'utente non potrà aggiungere, modificare o rimuovere direttamente elementi dall'elenco; per consentire all'utente di eseguire queste operazioni è necessario utilizzare altri controlli associandoli con del codice. Il controllo Elenco di voci è in grado di riconoscere e facilitare la modalità di selezione multipla di elementi, servendosi, per questa operazione dei tasti Maiusc o Ctrl .

Nella tabella seguente vengono indicate le proprietà più importanti di un controllo Elenco di voci.

Proprietà: sintassi	*Descrizione*	*Esempio*	*Commento*
List: `lstBox.`**`List`** `(indice) [= stringa]`	Restituisce o imposta le voci contenute in un elenco di voci.	`lstNames. List(0) = "Guglielmo"`	Assegna alla prima voce dell'elenco il nome `Guglielmo`.
ListCount: `lstBox.`**`ListCount`**	Restituisce il numero di elementi di un elenco di voci.	`If lstNames. ListCount > 0 Then lstNames. List(0) = "Guglielmo"`	Nel caso in cui il controllo non sia vuoto, sovrascrive il primo elemento di un controllo `"Elenco di voci"`.
ListIndex: `lstBox.`**`ListIndex`** `[= indice]`	Estrae o imposta l'indice della voce selezionata all'interno dell'elenco. Proprietà non disponibile in fase di progettazione del controllo. Quando non si effettua una selezione, a questa proprietà viene assegnato il valore `-1`.	`SelIdx = lstNames. ListIndex`	Memorizza nella variabile `SelIdx` l'indice corrispondente all'elemento selezionato dell'elenco.
Sorted: `lstBox.`**`Sorted`**	Determina se gli elementi di un elenco di voci appaiono ordinati.	`If lstNames. Sorted Then miaNuovaVoce "io" Else lstNames. AddItem "io"`	Esamina la proprietà `Sorted` per determinare se usare o meno una procedura personalizzata per inserire la voce "`io`" oppure se usare il metodo `AddItem( )` per inserire la medesima voce.

continua

Proprietà: sintassi	Descrizione	Esempio	Commento
Style: `lstBox.Style` *Nota.* Si tratta di una proprietà di sola lettura.	Determina lo stile di un elenco di voci. Il valore 0 (`vbListBoxStandard`) viene assunto per default e fa sì che il controllo si comporti come un comune elenco di voci. Il valore 1(`vbListBoxCheckbox`) fa sì che il controllo visualizzi un segno di spunta a fianco della voce selezionata.	`For Each Ctrl In mioElenco Voci If Ctrl.Style = 0 Then Ctrl.AddItem "io", 1 End If Next Ctrl`	Inserisce la stringa "io" a ogni elenco di voci che ha lo stile di default.
SelCount: `lstBox.SelCount`	Restituisce il numero delle voci selezionate.	`If lstBox.Sel Count = 1 Then mioProcesso End If`	Determina se un elenco di voci contiene una selezione di una sola voce e, quindi, la elabora.
Selected: `lstBox.Selected (indice) [= booleano]`	Restituisce o imposta lo stato di selezione di una voce all'interno di un elenco di voci.	`For I = 1 To lstBox. List Count lstBox.Selected(I) = True Next I`	Seleziona tutti gli elementi dell'elenco di voci `lstBox`.
MultiSelect: `lstBox.MultiSelect`	Restituisce un valore che indica se l'utente potrà effettuare una selezione multipla di elementi di un elenco di voci. Il valore 0 significa che l'elenco consente solo la selezione di una sola voce; 1 indica la possibilità di selezione multipla; 2 consente di attivare una selezione multipla estesa (ossia con la possibilità di selezionare voci non contigue). Si tratta di una proprietà di sola lettura.	`SelMode = lstNames. MultiSelect`	Memorizza il valore della proprietà `MultiSelect` di un elenco di voci e lo trasferisce alla variabile `SelMode`.

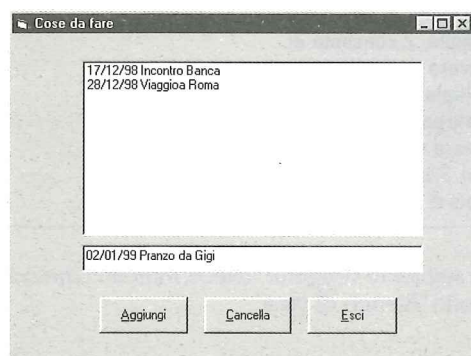
Nella tabella seguente vengono, invece, indicati i metodi associabili a un controllo "Elenco di voci".

Metodo: sintassi	Descrizione	Esempio	Commento
AddItem: `lstBox.AddItem voce, indice`	Inserisce un nuovo elemento in un elenco di voci.	`mioElenco.AddItem "io",`	Aggiunge la stringa "`io`" come primo elemento dell'elenco di voci.
Clear: `lstBox.Clear`	Svuota l'elenco di voci.	`lstNames.Clear`	Svuota il contenuto dell'elenco di voci `lstNames`.
RemoveItem: `lstBox.RemoveItem indice`	Rimuove dall'elenco di voci l'elemento specificato da indice.	`mioElenco. Remove 0`	Rimuove dall'elenco di voci `mioElenco` il primo elemento.

Nella tabella seguente vengono indicati gli eventi più importati associabili a un controllo Elenco di voci.

Evento	Descrizione
`Click`	Gestisce il clic all'interno dell'elenco di voci.
`DblClick`	Gestisce il doppio clic all'interno di un elenco di voci. Viene di solito utilizzato per eseguire un'operazione sull'elemento selezionato.
`Scroll`	Gestisce lo scorrimento del contenuto dell'elenco di voci.

Per maggiori informazioni controllo Elenco di voci, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

La figura seguente è il risultato di un programma che utilizza un controllo "Elenco di voci" per completare un semplice elenco di cose da fare.

Per inserire un nuovo elemento in un programma di voci l'utente dovrà scrivere il testo appropriato nella casella di immissione e quindi fare clic sul pulsante di comando Aggiungi. Per eliminare un elemento dall'elenco di voci, l'utente dovrà selezionare la voce da rimuovere e quindi fare clic sul pulsante di comando Cancella. Il programma, in questo caso, rimuoverà dall'elenco la voce selezionata riportandola all'interno della casella di immissione dalla quale, successivamente, potrà essere di nuovo inserita nell'elenco stesso.

Nella tabella seguente vengono riepilogati gli oggetti che fanno parte del modulo e le loro proprietà.

Oggetto	*Proprietà*	*Impostazione*
Form	Nome	Form1 (default)
	Caption	Cose da fare
Elenco di voci	Name	lstTasks
	Sorted	True
Casella di immissione	Name	txtTask
Pulsante comando	Name	cmdAdd
	Caption	&Aggiungi
Pulsante comando	Name	cmdDel
	Caption	&Cancella
Pulsante comando	Name	cmdQuit
	Caption	&Esci

Ecco il codice sorgente che genera la figura precedente.

```
Private Sub cmdAdd_Click( )
  Dim I As Integer
  If txtTasks.Text < > "" Then
    ' ricerca eventuali duplicati
    For I = 0 To lstTasks.ListCount -1
      If txtTasks.Text = lstTasks.List(I) Then
        Exit Sub
    Next I
    lstTasks.AddItem txtTasks.text
    txtTasks.Text = ""
  End If
End Sub
Private Sub cmdDel_Click( )
  Dim Indice As Integer
  Indice = lstTasks.ListIndex
```

```
    If Indice > -1 Then
      ' copia l'elemento selezionato
      ' nella casella di immissione
      txtTasks.text = lstTasks.List(Indice)
      ' rimuove l'elemento dall'elenco
      lstTasks.RemoveItem Indice
    End If
  End Sub
  Private Sub Form_Load( )
    txtTasks.Text ""
  End Sub
  Private Sub cmdQuit_Click( )
    End
  End Sub
```

La procedura `Form_Load( )` inizializza il form eliminando il testo dalla casella di immissione. La procedura `cmdQuit_Click( )` chiude il form e conclude l'applicazione.

La procedura `cmdAdd_Click( )` risponde a un clic da parte dell'utente sul pulsante di comando `Aggiungi` ed esegue le seguenti operazioni.

- Ricerca una voce nell'elenco che corrisponda al testo scritto nella casella di immissione. Questa operazione evita che l'utente immetta voci uguali a quelle già presenti nell'elenco. La routine utilizza un ciclo `For` per confrontare il contenuto dell'elenco all'interno dell'intervallo `0 To lstTasks.ListCount -1`. Il ciclo accede a ogni elemento dell'elenco servendosi della proprietà `List` e se una voce analizzata è uguale a quanto scritto nella casella di immissione, la routine viene conclusa senza inserire nell'elenco il contenuto della casella di immissione.
- Aggiunge all'elenco il contenuto della casella di immissione. Questa operazione si serve del metodo `AddItem( )`.
- Svuota dal contenuto la casella di immissione.

La procedura `cmdDel_Click( )` risponde a un'operazione di clic sul pulsante di comando `Cancella` eseguendo, quindi, le operazioni che seguono.

- Assegna alla variabile locale `Indice` la proprietà `ListIndex`.
- Determina se una voce è stata selezionata confrontando con `-1` il valore della variabile `Indice`. Se la variabile `Indice` contiene un valore superiore a `-1`, allora viene selezionato un elemento dell'elenco di voci e la procedura esegue le operazioni seguenti.

- Copia l'elemento selezionato nella casella di immissione. Questa operazione consente di accedere alla voce selezionata tramite la proprietà `List` e a un indice fornito dalla variabile `Indice`.
- Rimuove l'elemento selezionato servendosi del metodo `RemoveItem( )`. Il valore dell'argomento per richiamare questo metodo corrisponde alla variabile `Indice`.

Controllo "Pulsante opzione"

Il controllo Pulsante di opzione (Option Button) è composto da un piccolo elemento circolare che, quando l'opzione corrispondente viene attivata, contiene un pallino nero (in caso contrario, il pulsante appare vuoto). A fianco del pulsante di opzione viene di solito inserita un'etichetta descrittiva dell'opzione associata.

Tramite questo controllo si fornisce all'utente la possibilità di impostare l'opzione su "vero/falso" (o su "sì/no"). I pulsanti di controllo, quando fanno parte di un gruppo omogeneo (inserito di solito in un controllo "Riquadro"), si escludono l'uno con l'altro. Ciò significa che, all'interno dello stesso gruppo omogeneo, sarà possibile attivare solo un'opzione.

Le proprietà rilevanti di questo controllo sono `Value` e `Caption` e la sintassi generale per l'uso di `Value` è la seguente.

```
OptionControl.Value [= booleano]
```

La proprietà `Value` indica lo stato di un pulsante di opzione e i valori ammissibili per tale proprietà sono `False` (opzione disattivata) oppure `True` (opzione attivata). La proprietà `Value` può impostare o verificare lo stato di un pulsante di opzione.

Qui di seguito viene fornito un esempio di come sia possibile impostare un controllo Pulsante di opzione.

```
optBackup.Value = Not optBackup.Value
```

Questo esempio modifica lo stato della proprietà `Value` servendosi dell'operatore `Not`.

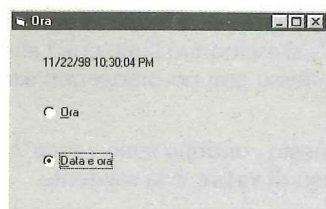
La proprietà `Caption` visualizza un testo descrittivo da associare a un controllo Pulsante di opzione. La sintassi generale per l'uso di questa proprietà è la seguente.

```
pulsanteOpzione.Caption [= stringa]
```

La proprietà `Caption` non è altro che una stringa che visualizza il testo descrittivo dell'opzione associata al pulsante e quello seguente è un esempio di come applicare correttamente tale proprietà.

```
Dim pulsanteOpzione, mieiControlli
For Each pulsanteOpzione In mieiControlli
If pulsanteOpzione.Caption = "Salvataggio automatico" Then
  pulsanteOpzione.Value = False ' svuota il pulsante
  Exit For ' Esce dal ciclo
End If
Next pulsanteOpzione
```

L'esempio utilizza un ciclo `For Each` per ricercare il controllo Pulsante di opzione nel quale la proprietà `Caption` è impostata su Salvataggio automatico. Quando il ciclo reperisce il pulsante di opzione cercato, imposta la proprietà `Value` su `False`.

La figura seguente è il risultato di un esempio che prevede l'uso di un controllo Pulsante di opzione che visualizza l'ora e la data di sistema.

Inizialmente, il form visualizza solo l'ora corrente e quando l'utente farà clic sul pulsante di opzione `Data e Ora`, nel modulo stesso, oltre all'ora, verrà visualizzata anche la data di sistema. Quando, invece, l'utente farà clic sul pulsante di opzione `Ora`, nel form verrà visualizzata solo l'ora. Nella tabella seguente si riportano le impostazioni dei vari elementi del modulo generato.

Oggetto	*Proprietà*	*Impostazione*
Modulo	Name	Form1 (default)
	Caption	Ora
Etichetta	Name	lblTime
Temporizzatore (Timer)	Name	tmrTime
	Interval	1000
Pulsante opzione	Name	optTime
	Caption	&Ora
Pulsante opzione	Name	optTimeDate
	Caption	&Data e Ora

Quello che segue è il codice sorgente che gestisce il form.

```
Option Explicit
Private Sub ShowTime( )
  Dim TStr As Variant
  TStr = Time
  If optTimeDate.Value = True Then
    TStr = TStr + Date
  End If
  lblTime.Caption = TStr
End Sub
Private Sub Form_Load( )
  lblTime.Caption = Time
End Sub
Private Sub optTime_Click( )
  ShowTime
End Sub
Private Sub optTimeDate_Click( )
  ShowTime
End Sub
Private Sub tmrTime_Timer( )
  ShowTime
End Sub
```

Da questo codice sorgente risultano evidenti i seguenti gestori di evento.

- La procedura `Form_Load( )`, all'avvio dell'applicazione, visualizza nell'etichetta del modulo l'ora.
- La procedura `optTimer_Click( )` risponde a un clic dell'utente sul pulsante di opzione `Ora`. Questa procedura richiama la procedura `ShowTime( )` per aggiornare la visualizzazione, nell'etichetta del modulo, della data/ora di sistema.
- La procedura `tmrTime_Timer( )` richiama, ogni secondo, la procedura `ShowTime( )` in modo da aggiornare costantemente l'etichetta del modulo che riporta la data/ora di sistema.
- La procedura `ShowTime( )`, nel caso in cui l'opzione `Ora` risulti attiva, assegna l'ora corrente di sistema alla proprietà `Caption` dell'etichetta del modulo. Al contrario, se risulta selezionata l'opzione `Data e Ora`, la procedura include, nella proprietà `Caption` dell'etichetta del modulo, il valore corrispondente alla data di sistema.

Controllo "Pulsanti Radio"

Si veda il controllo "Pulsanti Opzione".

Controllo "Barra di scorrimento"

Visual Basic consente di creare anche i controlli Barra di scorrimento verticale (Vertical Scroll Bar) e Barra di scorrimento orizzontale (Horizontal Scroll Bar). A parte le ovvie considerazioni sull'orientamento, questi due controlli sono molto simili poiché consentono entrambi all'utente di visualizzare e selezionare rapidamente un valore contenuto in un ampio intervallo di valori. I controlli per le barre di scorrimento orizzontale o verticale possono anche fornire una rappresentazione della posizione corrente di un valore particolare all'interno del programma. Nella tabella seguente vengono descritte le proprietà rilevanti dei controlli per le barre di scorrimento.

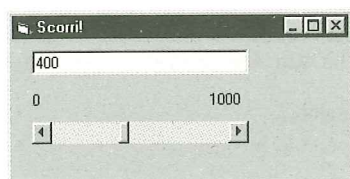
Proprietà: sintassi	*Descrizione*	*Esempio*	*Commento*
Min: `scrBar.`**`Min`** `[= valore]`	Restituisce o imposta il valore minimo di una barra di scorrimento.	`scrTime.Min = 0`	Imposta a 0 la proprietà `Min`.
Max: `scrBar.`**`Max`** `[= valore]`	Restituisce o imposta il valore massimo di una barra di scorrimento.	`scrTime.Max = 100`	Imposta a 100 il valore della proprietà `Max`.
SmallChange: `scrBar.`**`SmallChange`** `[= valore]`	Restituisce o imposta l'intervallo di modifica della proprietà Value di una barra di scorrimento quando l'utente farà clic su una delle frecce di scorrimento poste agli estremi delle barre.	`scrTime.Small Change = 1`	Imposta la proprietà `Small Change` a 1.
LargeChange: `scrBar.`**`LargeChange`** `[= valore]`	Restituisce o imposta la modifica della proprietà Value di una barra di scorrimento quando l'utente farà clic nelle aree poste tra le frecce di scorrimento poste alle estremità delle barre stesse e il pulsante di scorrimento.	`scrTime.Large Change = 10`	Imposta a 10 la proprietà `LargeChange`.

continua

Proprietà: sintassi	Descrizione	Esempio	Commento
Value: `scBar.Value [ = value]`	Restituisce o imposta la posizione corrente della barra di scorrimento e il valore restituito è sempre contenuto nell'intervallo di valori definiti dalle proprietà Min e Max.	`scrTime.Value = (scrTime. Min + scrTime.Max) / 2`	Imposta la proprietà `Value` a un valore medio di quelli definiti dalle proprietà `Min` e `Max`.

Nella tabella seguente vengono, invece, descritti i principali eventi associabili alle barre di scorrimento.

Evento	Descrizione
`Scroll`	Gestisce lo scorrimento della barra.
`Change`	Gestisce la modifica del valore della barra di scorrimento.

La figura seguente è un esempio dell'utilizzo di una barra di scorrimento orizzontale che consente di selezionare un numero compreso tra 0 e 1000.

Quando l'utente modificherà il valore dalla barra di controllo, questo verrà riflesso anche nella casella di immissione. In modo analogo, quando l'utente scriverà un valore all'interno della casella di immissione, tale valore verrà riflesso anche a livello della barra di scorrimento. Nella tabella seguente vengono riportate le impostazioni per i controlli e per il modulo della figura precedente.

Oggetto	Proprietà	Impostazione
Form	Name Caption	Form1 (default) Scorri!
Etichetta	Name	lblMin
Etichetta	Name	lblMax
Casella di modifica	Name	txtValue
Barra scorrimento	Name	scrValue

Quello che segue è il codice sorgente per l'applicazione.

```
Option Explicit
Const MIN_VAL = 0
Const MAX_VAL = 1000

Private Sub Form_Load( )
  ' inizializza la barra di scorrimento
  scrValue.Min = MIN_VAL
  scrValue.Max = MAX_VAL
  scrValue.SmallChange = 1
  scrValue.LargeChange = (MAX_VAL - MIN_VAL) / 10
  scrValue.Value = MIN_VAL
  ' inizializza gli altri controlli
  lblMin.Caption = Str(MIN_VAL)
  lblMax.Caption = Str(MAX_VAL)
  txtValue.Text = Mid(Str(MIN_VAL), 2)
End Sub
Private Sub scrValue_Change( )
  txtValue.Text = Mid(Str(scrValue.Value), 2)
End Sub
Private Sub scrValue_Scroll( )
  txtValueText = Mid(Str(scrValue.Value), 2)
End Sub
Private Sub txtValue_Change( )
  Dim StrVal As Integer
  StrVal = Val(txtValue.Text)
  If StrVal >= scrValue.Min And StrVal <= scrValue.Max
Then
    scrValue.Value = StrVal
  End If
End Sub
```

In questo codice si dichiarano le costanti `MIN_VAL` e `MAX_VAL` che definiscono l'intervallo dei valori ammissibili (`0` e `1000`) nella barra di scorrimento. La procedura `Form_Load( )` inizializza le proprietà più importanti del controllo Barra di scorrimento,il testo delle etichette e quello degli altri controlli ed esegue le seguenti operazioni.

- Assegna alla proprietà `Min` del controllo Barra di scorrimento il valore della costante `MIN_VAL`.
- Assegna alla proprietà `Max` della Barra di scorrimento il valore della costante `VAL_MAX`.
- Assegna alla proprietà `SmallChange` della Barra di scorrimento il valore 1.
- Assegna alla proprietà `LargeChange` della barra di scorrimento l'espressione (`MAX_VAL - MIN_VAL`) / 10.

- Assegna alla proprietà `Value` della Barra di scorrimento il valore della costante `MIN_VAL`.
- Assegna all'etichetta `lblMin` il valore della costante `MIN_VAL` convertito in stringa. Questa operazione visualizza "0", che corrisponde al valore minimo dell'intervallo di valori per la Barra di scorrimento.
- Assegna all'etichetta `lblMax` il valore della costante `MAX_VAL` convertito in stringa. Questa operazione visualizza "1000", che corrisponde al valore massimo dell'intervallo di valori per la Barra di scorrimento.
- Assegna alla casella di modifica il valore della costante `MIN_VAL` convertito in stringa. Questa operazione fa sì che nella casella di immissione appaia "0", corrispondente al valore corrente della Barra di scorrimento.

Inoltre, questo codice contiene i seguenti eventi.

- La procedura `scrValue_Change( )` risponde all'evento `Change` della Barra di scorrimento e scrive la stringa corrispondente nella casella di immissione.
- La procedura `scrValue_Scroll( )` risponde all'evento `Scroll` della Barra di scorrimento e scrive all'interno della casella di modifica l'immagine della stringa del valore raggiunto dalla barra di scorrimento.
- La procedura `txtValue_Change( )` risponde all'evento `Change` della casella di immissione e converte il testo immesso nella casella stessa in un valore numerico intero. A questo punto il valore viene assegnato alla barra di scorrimento sempre che questo sia compreso nell'intervallo di valori definito dalle proprietà `Min` e `Max` della barra di scorrimento stessa.

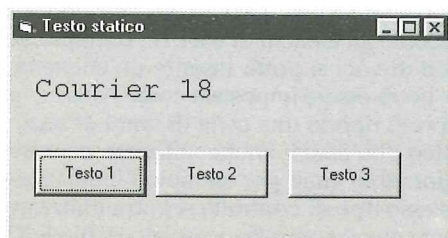
Controllo "Testo Statico"

Il controllo Testo statico (Label), definito anche *etichetta*, visualizza un testo per quei controlli che, di solito, non prevedono la presenza di un'etichetta, come gli elenchi di voci. Per esempio, accanto a un controllo Elenco di voci si potrà inserire un'etichetta la cui proprietà `Caption` potrà essere impostata come `Paesi:`, per indicare che l'elenco di voci riporta una serie di nomi di nazioni. Un controllo Testo statico può essere anche utilizzato per visualizzare informazioni aggiornabili quali, per esempio, l'indicazione della cartella attiva. Lo stesso tipo di controllo si potrà utilizzare anche per informare l'utente che il controllo associato richiede l'immissione di un testo o l'avvio di un'operazione particolare. Di solito, un con-

trollo Testo statico non viene definito per rispondere a eventi dell'utente anche se si potrà definirlo perché risponda a operazioni quali clic o spostamenti del mouse.

Nella tabella seguente vengono visualizzate le proprietà più importanti che si possono associare al controllo Testo statico.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Caption: `oggetto.`**`Caption`** `[= stringa]`	Restituisce o imposta il testo di un controllo Testo statico.	`lblNomeFile. Caption = "Nome file:"`	Imposta su Nome file: la proprietà Caption di un'etichetta.
ForeColor: `oggetto.`**`ForeColor`** `[= numero colore]`	Restituisce o imposta il colore del testo di un'etichetta.	`lblText. Fore Color = vbRed`	Imposta il colore del testo dell'etichetta in modo che appaia in rosso.
BackColor: `oggetto.`**`BackColor`** `[= numero colore]`	Restituisce o imposta il colore di sfondo di un'etichetta.	`lblText. Back Color = vbRed`	Imposta sul rosso il colore di sfondo di un'etichetta.
FontName: `oggetto.`**`Fontname`** `[= stringa]`	Restituisce o imposta il tipo di carattere (*font*) da applicare al testo dell'etichetta.	`lblText. FontName = "Arial"`	Imposta su "Arial" il tipo di carattere da applicare al testo dell'etichetta.
FontSize: `oggetto.`**`FontSize`** `[= numero]`	Restituisce o imposta la dimensione del carattere da applicare al testo dell'etichetta.	`lblText. Font Size = 24`	Imposta a 24 la dimensione del carattere da applicare al testo dell'etichetta.

La figura seguente mostra un esempio del risultato di un'applicazione che utilizza controlli di tipo Testo statico. L'esempio visualizza un controllo Testo statico e tre pulsanti di comando, ognuno dei quali imposta nuovi valori per il testo, il tipo di carattere e la sua dimensione all'interno dell'etichetta.

Nella tabella seguente si riportano le proprietà degli elementi contenuti nella finestra.

Oggetto	*Proprietà*	*Impostazione*
Form	Name	Form1 (default)
	Caption	Etichette
Etichetta	Name	lblText
Pulsante comando	Name	cmdText1
	Caption	Testo 1
Pulsante comando	Name	cmdText2
	Caption	Testo 2
Pulsante comando	Name	cmdText3
	Caption	Testo 3

Di seguito viene presentato il codice sorgente dell'applicazione.

```
Option Explicit
Private Sub cmdText1_Click( )
  lblText.Caption = "Arial 24"
  lblText.FontName = "Arial"
  lblText.FontSize = 24
  lblText.ForeColor = vbRed
End Sub
Private Sub cmdText2_Click( )
  lblText.Caption = "Courier 18"
  lblText.FontName = "Courier"
  lblText.FontSize = 18
  lblText.ForeColor = vbBlue
End Sub
Private Sub cmdText2_Click( )
  lblText.Caption = "Times Roman 20"
  lblText.FontName = "Times Roman"
  lblText.FontSize = 20
  lblText.ForeColor = vbYellow
End Sub
```

La procedura `cmdText1_Click( )` risponde al clic dell'utente sul pulsante di comando `Testo 1` eseguendo le operazioni seguenti.

- Assegna alla proprietà `Caption` dell'etichetta il valore `Arial 24`.
- Assegna alla proprietà `FontName` dell'etichetta il valore `Arial`.
- Assegna alla proprietà `FontSize` dell'etichetta il valore `24`.
- Assegna alla proprietà `ForeColor` dell'etichetta il valore `vbRed` (testo in rosso).

La procedura `cmdText2_Click( )` risponde al clic dell'utente sul pulsante di comando `Testo 2` eseguendo le operazioni seguenti.

- Assegna alla proprietà `Caption` dell'etichetta il valore `Courier 18`.
- Assegna alla proprietà `FontName` dell'etichetta il valore `Courier`.
- Assegna alla proprietà `FontSize` dell'etichetta il valore `18`.
- Assegna alla proprietà `ForeColor` dell'etichetta il valore `vbBlue` (testo in blu).

La procedura `cmdText3_Click( )` risponde al clic dell'utente sul pulsante di comando `Testo 3` eseguendo le operazioni seguenti.

- Assegna alla proprietà `Caption` dell'etichetta il valore `Times Roman 20`.
- Assegna alla proprietà `FontName` dell'etichetta il valore `Times Roman`.
- Assegna alla proprietà `FontSize` dell'etichetta il valore `20`.
- Assegna alla proprietà `ForeColor` dell'etichetta il valore `vbYellow` (testo in giallo).

Controllo "Casella di testo"

Il controllo Casella di testo (Text Box), definito anche *casella di immissione,* consente di immettere informazioni sia in fase di progettazione sia in fase di esecuzione dell'applicazione.

Nella tabella seguente si riportano le proprietà più importanti del controllo Casella di testo. Gli eventi più rilevanti per questo tipo di controllo sono, invece, `Click`, `GotFocus`. `LostFocus`, `Change` ed altri eventi collegati alla pressione di alcuni tasti.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Text: `oggetto.`**`Text`** `[= stringa]`	Restituisce o imposta il testo della casella di immissione.	`txtName.Text = "Marzia"`	Memorizza nella proprietà `Text` della casella di testo il nome "`Marzia`".
Enabled: `oggetto.`**`Enabled`** `[= booleano]`	Restituisce o imposta lo stato di abilitazione per la casella di testo.	`txtName.Enabled = False`	Disabilita l'immissione di testo nella casella di immissione.
Visible: `oggetto.`**`Visible`** `[= booleano]`	Restituisce o imposta la visibilità della casella di testo.	`txtName.Visible = True`	Nasconde la casella di testo.

continua

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
MultiLine: `oggetto.MultiLine`	Restituisce la modalità di casella di testo su più righe.	`If Not txtName.MultiLine Then numLines = 1End If`	Se la proprietà `MultiLine` è falsa, imposta la variabile `numLines` su 1.

Per maggiori informazioni sul controllo Casella di testo, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

La figura seguente è il risultato di un'applicazione che utilizza controlli Casella di testo.

Il modulo presenta le seguenti caselle di immissione.

- `Maiuscole` che accetta caratteri alfabetici e li converte automaticamente in lettere maiuscole.
- `Minuscole` che accetta caratteri alfabetici e li converte automaticamente in lettere minuscole.
- `Cifre` che accetta l'immissione solo di numeri.
- `Qualsiasi testo` che accetta qualsiasi carattere alfanumerico e prevede una possibilità di sviluppo su più righe.

Nella tabella seguente vengono riportate le impostazioni delle proprietà dei controlli e del form stesso.

Oggetto	*Proprietà*	*Impostazione*
Form	Name	Form1 (default)
	Caption	Demo Casella di testo
Etichetta	Name	lblUpperCase
	Caption	Maiuscole
Etichetta	Name	lblLowerCase
	Caption	Minuscole
Etichetta	Name	lblDigits
	Caption	Cifre
Label	Name	lblAnyText
	Caption	Qualsiasi testo
Casella testo	Name	txtUpperCase
	Text	(vuoto)
Casella testo	Name	txtLowerCase
	Text	(vuoto)
Casella testo	Name	txtDigits
	Text	(vuoto)
Casella testo	Name	txtAnyText
	Text	(vuoto)
	MultiLine	True

Ecco il codice sorgente per l'applicazione.

```
Private Sub txtAnyText_GotFocus( )
  If txtAnyText.Text = " " Then
    txtAnyText.Text = "Ciao!"
  End If
End Sub
Private Sub txtDigits_KeyPress(KeyAscii As Integer)
  If InStr("0123456789", Chr(KeyAscii)) = 0 Then
    KeyAscii = 0
  End If
End Sub
Private Sub txtLowerCase_KeyPress(KeyAscii As Integer)
  If KeyAscii >= Asc("a") Or KeyAscii <= Asc("Z") Then
    KeyAscii = KeyAscii + Asc("a") - Asc("A")
  ElseIf KeyAscii >= Asc("a" Or KeyAscii <= Asc("z") Then
    Exit Sub
  Else
    KeyAscii = 0
  End If
End Sub
Private Sub txtUpperCase_KeyPress(KeyAscii As Integer)
```

```
  If KeyAscii >= Asc("a" Or KeyAscii <= Asc("z") Then
    KeyAscii = KeyAscii - Asc("a") + Asc("A")
  ElseIf KeyAscii >= Asc("A") Or KeyAscii <= Asc("Z")
Then
    Exit Sub
  Else
    KeyAscii = 0
  End If
End Sub
```

La procedura `txtAnyText_GotFocus( )` risponde all'evento `GotFocus` della casella di testo `AnyText`. La routine utilizza un enunciato `If` per determinare se la casella non contiene alcun testo. Quando questa combinazione risulta vera, la routine scrive nella casella `Any Text` la stringa `Ciao`!.

La procedura `txtDigits_KeyPress( )` risponde all'operazione di immissione nella casella `Cifre`. La procedura prevede l'argomento `KeyAscii` di tipo `Integer`, che rappresenta il codice ASCII (*American Standard Code for Information Interchange*) per il carattere immesso. La routine usa un enunciato `If` per determinare se il carattere digitato corrisponde a un numero. La condizione dell'enunciato `If` richiama la funzione `InStr( )` e confronta il risultato della funzione con il valore 0. Quando questa condizione risulta vera (ovvero quando l'utente ha inserito un carattere non numerico) il gestore di evento assegna all'argomento `KeyAscii` il valore "0". Questo valore ignora, sostanzialmente, le immissioni dell'utente e le invia a un "buco nero".

La procedura `txtLowerCase_KeyPress( )` risponde alle immissioni dell'utente nella casella di testo `Maiuscole` verificando le condizioni seguenti tramite un enunciato `If` con più alternative.

- Quando il carattere immesso dall'utente è maiuscolo, il gestore di evento lo converte in minuscolo modificando il valore dell'argomento `KeyAscii`.
- Se il carattere immesso è in minuscolo, l'evento si interrompe poiché il valore dell'argomento `KeyAscii` è corretto (il carattere immesso è minuscolo).
- Quando il carattere immesso non è alfabetico, il gestore di evento assegna all'argomento `KeyAscii` il valore `0`, ignorando così l'immissione.

La procedura `txtUppercase_KeyPress( )` è gestore di evento e risponde alle immissioni dell'utente nella casella di testo `Upper Case` verificando le condizioni seguenti tramite un enunciato `If` con più alternative.

- Quando il carattere immesso dall'utente è minuscolo, il gestore di evento lo converte in maiuscolo modificando il valore dell'argomento `KeyAscii`.
- Se il carattere immesso è maiuscolo, l'evento si interrompe poiché il valore dell'argomento `KeyAscii` è corretto (il carattere immesso è maiuscolo).
- Quando il carattere immesso non è alfabetico, il gestore di evento assegna all'argomento `KeyAscii` il valore `0`, ignorando così l'immissione.

Parte III

Classi e tipi di dati

In questa parte vengono presi in considerazione argomenti quali la creazione e l'uso delle classi e dei tipi di dati. Vengono anche prese in esame variabili e costanti e si descriverà come utilizzare i vari operatori per confrontare i tipi di dati, le variabili e le costanti.

Argomenti trattati

- **Creazione e uso delle classi**
- **I vari tipi di dati di Visual Basic**
- **Definizione di tipi di dati enumerati e di altri tipi di dati**
- **Uso degli operatori Visual Basic**
- **Variabili e costanti**

Creazione di tipi di dati personalizzati

Visual Basic permette di dichiarare propri tipi di dati personalizzati servendosi, per questa operazione, dell'enunciato `Type`. L'uso di questo enunciato all'interno di un modulo consente di dichiarare un tipo di dati personalizzato contenente uno o più elementi.

Vedi anche: in questa parte, la sezione che descrive l'uso dei tipi di dati personalizzati.

La sintassi generale per dichiarare un tipo di dati personalizzato è la seguente.

```
[Private ¦ Public] Type nomeTipo
  nomeElemento [([subscripts])] As tipo
  [nomeElemento [([subscripts])] As tipo]
  ...
End Type
```

La parola chiave facoltativa `Public` fa sì che il tipo di dati personalizzato sia disponibile per tutte le procedure di tutti i moduli del progetto. La parola chiave `Private`, al contrario, indica che il tipo di dati personalizzato è disponibile solo all'interno del modulo nel quale il tipo di dati è stato dichiarato. La voce `nomeTipo` che compare dopo la parola chiave `Type`, corrisponde al nome assegnato al tipo di dati personalizzato e deve obbedire alle convenzioni standard per la creazione dei nomi in Visual Basic.

La voce `nomeElemento` corrisponde al nome di un elemento del tipo di dati personalizzato e deve anch'essa assoggettarsi alle convenzioni standard per l'assegnazione di nomi di variabili, con la sola eccezione che è possibile anche utilizzare parole chiave. La parte `subscripts` specifica le dimensioni di un elemento matrice (*array*). Per dichiarare una matrice di dimensione variabile si utilizzino solo le parentesi tonde senza racchiudervi alcunché.

L'argomento `subscripts` prevede la sintassi seguente.

```
[min To] max [,[min To] max] ...
```

Quando si omette di scrivere la parte `min`, il limite inferiore della matrice viene controllato dall'enunciato `Option Base`. Per default, se non si utilizza l'enunciato `Option Base`, il limite inferiore è pari a `0`.

`tipo` specifica il tipo di dati dell'elemento e può essere pari a `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (per le stringhe di lunghezza variabile), `String * lunghezza` (per le stringhe a lunghezza fissa), `Object`, `Variant` o un qualsiasi altro tipo di dato o di oggetto personalizzato.

Per maggiori informazioni circa la creazione di tipi di dati personalizzati, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello che segue è un esempio di come impostare un tipo di dati personalizzato.

```
Type DatiGioco
  IniziaGioco As Boolean
  NumeroIterazioni As Integer
  MaxIterazioni As Integer
  Segreto As Byte
  Indovina As Byte
  Messaggio As String
End Type
```

Questo esempio dichiara il tipo di dati personalizzato `DatiGioco` nel quale memorizzare le informazioni in un ipotetico gioco per indovinare un numero segreto.

Il tipo di dati personalizzato contiene i seguenti elementi.

- L'elemento `IniziaGioco` è di tipo `Boolean` e serve come indicatore per verificare se il gioco è o meno ancora in corso.
- L'elemento `NumeroIterazioni` è di tipo `Integer` e memorizza il numero di tentativi per indovinare un numero segreto.
- L'elemento `MaxIterazioni` è di tipo `Integer` e memorizza il numero massimo dei tentativi ammissibili per indovinare il numero segreto.
- L'elemento `Segreto` è di tipo `Byte` e contiene il numero segreto da indovinare.
- L'elemento `Indovina` è di tipo `Byte` e memorizza il numero che si presume sia quello segreto.
- L'elemento `Messaggio` è di tipo `String` e memorizza il messaggio che dà informazioni nel corso della sessione di gioco.

Quello che segue è un altro esempio di come definire un tipo di dati personalizzato: questa volta vengono usati elementi che sono delle matrici.

```
Type MiniFoglioCalcolo
  NumRighe As Byte
  NumCol As Byte
  Celle(50, 50) As Double
  SommaCol(50) As Double
```

```
  SommaRighe(50) As Double
  SegnalaErrore As Boolean
  MessaggioErrore As String
End Type
```

Questo esempio dichiara il tipo di dati personalizzato `MiniFoglio-Calcolo` per memorizzare le informazioni contenute in un foglio di calcolo di dimensioni molto limitate. Il tipo di dati personalizzato prevede i seguenti elementi.

- L'elemento `NumRighe` è di tipo `Byte` e memorizza il numero delle righe contenenti dati.
- L'elemento `NumCol` è di tipo `Byte` e memorizza il numero delle colonne contenenti dati.
- L'elemento `Celle` è di tipo `Double` e memorizza i dati in una matrice bidimensionale composta da 51 righe e 51 colonne.
- L'elemento `SommaRighe` è di tipo `Double` e memorizza la somma delle righe di una matrice monodimensionale composta da 51 elementi.
- L'elemento `SommaCol` è di tipo `Double` e memorizza la somma delle righe di una matrice monodimensionale composta da 51 elementi.
- L'elemento `SegnalaErrore` è di tipo `Boolean` e corrisponde a una indicazione di errore.
- L'elemento `MessaggiErrore` è di tipo `String` e memorizza i messaggi di errore.

Creazione di tipi enumerati personali

Visual Basic, tramite l'enunciato `Enum`, permette di dichiarare un tipo di enumerazione. Questo tipo di dati particolare permette di elencare una serie di costanti che descrivono i vari stati o valori di qualche oggetto. Per esempio, si potrà dichiarare un tipo `Enum` che elenchi i giorni della settimana.

Vedi anche: in questa parte, la sezione che descrive l'uso di tipi enumerati personali.

La sintassi generale per l'enunciato `Enum`, è la seguente.

```
[Public ¦ Private] Enum nome
  nomemembro [= espressioneCostante]
  nomemembro [= espressioneCostante]
  ...
End Enum
```

La parola chiave facoltativa `Public` indica che il tipo `Enum` sarà disponibile per tutto il progetto. Per default, i tipi `Enum` sono `Public`. La parola chiave facoltativa `Private`, al contrario, indica che il tipo `Enum` sarà disponibile solo all'interno del modulo che contiene il tipo di dati stesso.

La parte `nome` specifica il nome del tipo `Enum` e deve seguire le convenzioni standard per l'assegnazione dei nomi. Visual Basic riconosce la parte `nome` come un tipo di dati solo quando vengono dichiarate le variabili o i parametri del tipo `Enum`. La parte nomemembro corrisponde al nome in base al quale viene riconosciuto l'elemento costitutivo del tipo di dati `Enum`.

La parte `espressioneCostante` è un valore facoltativo dell'elemento (esegue una valutazione fino al tipo `Long`). Il valore corrispondente può essere specificato in un altro tipo `Enum` e se si omette la parte `espressioneCostante`, Visual Basic assegnerà il valore `0` (se si tratta del primo `nomemembro`) o `1` più il valore di `nomemembro` immediatamente precedente.

Viene ora fornito un esempio di un tipo di dati `Enum`.

```
Enum Giorni
  GiornoNonValido = 0
  Lunedì
  Martedì
  Mercoledì
  Giovedì
  Venerdì
  Sabato
  Domenica
End Enum
```

Questo esempio dichiara `Giorni` come tipo di dati `Enum`. Il primo membro del tipo di dati corrisponde a `GiornoNonValido` e allo stesso viene esplicitamente assegnato il valore 0. Il secondo membro è `Lunedì` senza associazione ad alcun valore esplicito. Visual Basic, in questo caso, assegnerà a `Lunedì` il valore 1 (che corrisponde al valore di `GiornoNonValido` [0] incrementato di 1). Il tipo `Enum` contiene altri nomi di membri corrispondenti agli altri giorni della settimana ai quali Visual Basic assegnerà, rispettivamente, i valori da 2 a 7. Poiché è stato omesso l'argomento `espressioneCostante`, Visual Basic assegnerà automaticamente a ogni giorno della settimana il numero del membro precedente, incrementato di 1.

Tipi di dati di Visual Basic

Visual Basic è dotato di una serie di tipi di dati predefiniti elencati nella tabella seguente nella quale si indica anche il nome, la dimensione di memorizzazione e l'intervallo di azione di ogni tipo di dati.

Tipo di dati	*Dimensione*	*Intervallo memorizzazione*
`Byte`	1 byte	da 0 a 255
`Boolean`	2 byte	True o False
`Integer`	2 byte	Da –32.768 a 32.767
`Long` (intero lungo)	4 byte	Da –2.147.483.648 a 2.147.483.647
`Single` (virgola mobile semplice)	4 byte	Da –3,402823E38 a –1,401298E-45 per la precisione dei valori negativi e da 1,401298E-45 a 3,402823E38 per la precisione dei numeri positivi.
`Double` (virgola mobile a doppia precisione)	8 byte	–4,94065645841247E-324 per i valori negativi e 4.940656458 41247E-324 per i valori positivi.
`Currency` (interi scalari)	8 byte	Da –922.337.203. 685. 477,5808 a 922.337. 203.685.477,5807".
`Decimal`	14 byte	+/–79.228.162.514.264.337.593.543.950.335 senza cifra decimale e +/–7,9228162514264337593543950335 con 28 cifre decimali dopo la virgola. Il valore minimo diverso da zero è pari a +/– 0,0000000000000000000000000001.
`Date`	8 byte	Dal 1 gennaio 100 al 31 dicembre 9999.
`Object`	4 byte	Qualsiasi riferimento a oggetto.
`String`	10 byte + lunghezza stringa	Da 0 ad, approssimativamente, 2 miliardi.
`String`	Lunghezza della stringa	da 1 ad, approssimativamente, 65.400.
`Variant` (con numeri)	16 byte	Un qualsiasi valore numerico fino all'intervallo di `Double`.
`Variant`	22 byte + lunghezza stringa	Da 0 ad, approssimativamente, 2 miliardi.

continua

Tipo di dati	*Dimensione*	*Intervallo memorizzazione*
Personalizzato (definito da `Type`)	Numero richiesto dagli elementi	L'intervallo di ogni elemento è identico a quello dell'intervallo per il tipo di dati.

Dichiarazione classi

Visual Basic consente di dichiarare una classe che specifichi gli elementi di dati e i metodi relativi. I progetti Visual Basic memorizzano le classi in file separati con estensione ".CLS". Per aggiungere un file classe si avvii il comando Progetto ⇨ Inserisci modulo di classe.

Gli elementi dati che vengono specificati in una classe possono essere tipi di dati predefiniti, personalizzati, enumerati personali, o altre classi.

Vedi anche: in questa parte, la sezione che descrive come usare le classi.

La sintassi generale per dichiarare un elemento dati è la seguente.

```
[Private ¦ Public] elementoDati As tipo
```

La parola chiave `Private` dichiara un elemento dati che sarà accessibile solo alle routine della medesima classe. Al contrario, la parola chiave `Public` rende disponibili gli elementi dati anche dall'esterno del modulo classe.

- Molti programmatori consigliano di dichiarare gli elementi classe come `Private` in modo da poter controllare l'accesso a tali elementi. Le funzioni e le procedure della classe possono regolare l'accesso agli elementi dati per assicurare che le informazioni contenute negli elementi dati stessi non vengano danneggiate da accessi non appropriati provenienti da altri moduli.
- Molti programmatori consigliano di dichiarare le routine come `Private` tutte le volte che queste operano con altre routine della stessa classe.

Si ha la possibilità di dichiarare funzioni e procedure in modo che queste si comportino come metodi che siano in grado di manipolare elementi dati interni alla classe. Una routine può essere dichiarata come `Public` o `Private`. Nel primo saranno accessibili ai componenti di altre classi o altri moduli mentre le routine dichiarate come `Private` potranno operare solo con altre routine della stessa classe.

Per maggiori informazioni sulle classi, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Ecco un esempio di codice per la dichiarazione di una classe.

```
Option Explicit
Private mintXCoord As Integer
Private mintXCoord As Integer

Public Sub MoveTo(X As Integer, Y As Integer)
  mintXCoord = X
  mintYCoord = Y
End Sub
Public Sub MoveBy(XDist As Integer, YDist As Integer)
  mintXCoord = mintXCoord + XDist
  mintYCoord = mintYCoord + YDist
End Sub
Function GetX( ) As Integer
  GetX = mintXCoord
End Function
Function GetY( ) As Integer
  GetY = mintYCoord
End Function
Public SubMove ToCoord(Coord As CPoint)
  mintXCoord = Coord.GetX
  mintYCoord = Coord.GetY
End Sub
```

In questo codice si dichiara la classe `Point` che prevede i seguenti elementi dati.

- L'elemento dati `mintXCoord` di tipo `Private Integer` che memorizza il valore della coordinata `X`.
- L'elemento dati `mintYCoord` di tipo `Private Integer` che memorizza il valore della coordinata `Y`.

La classe dichiara inoltre le routine `MoveTo( )`,`MoveBy( )`,`GetX( )`, `GetY( )` e `MoveToCoord( )`. La procedura `MoveTo( )` prevede gli argomenti `X` e `Y` di tipo `Integer` che specificano le nuove coordinate. La procedura copia i valori di questi argomenti negli elementi dati `mintXCoord` e `mintYCoord`.

La procedura `MoveBy( )` prevede gli argomenti `XDist` e `YDist` di tipo `Integer` che specificano lo spostamento delle coordinate `X` e `Y`. La procedura aggiunge i valori di questi argomenti agli elementi dati `mintXCoord` e `mintYCoord`. Le funzioni `GetX( )` e `GetY( )` restituiscono i valori degli elementi dati `mintXCoord` e `mintYCoord`.

La procedura `MoveToCoord( )` copia le coordinate dell'argomento `Coord` di tipo `CPoint` (che è un'istanza della classe `CPoint`) negli elementi dati `mintXCoord` e `mintYCoord`. Si osservi che la procedura accede agli elementi dati degli argomenti `Coord` servendosi delle funzioni `GetX( )` e `GetY( )` poiché gli elementi dati sono stati definiti come `Private`.

Uso degli operatori aritmetici

Visual Basic prevede la presenza di una serie di operatori aritmetici i cui singoli significati vengono descritti nella tabella seguente. Questi operatori consentono di eseguire operazioni di calcolo su dati numerici.

Operatore	*Sintassi*	*Commenti*	*Esempio*
+	`espressione1 + espressione2`	Esegue la somma di due valori.	`23 + 45` che genera il risultato 68.
–	`espressione1 - espressione 2`	Esegue la sottrazione di due valori.	`100 - 45` che genera il risultato 55.
*	`espressione1 *espressione2`	Esegue la moltiplicazione di due valori.	`0.34 * 23.45` che genera il risultato 7.9730.
/	`espressione 1 / espressione2`	Esegue la divisione di due valori per ottenere un risultato in virgola mobile.	`355.2 / 113.1` che genera il risultato 3.1406.
\	`espressione1 \ espressione2`	Esegue una divisione di due valori senza cifre decimali.	`355 \ 113` che genera il risultato 3.
^	`espressione1 ^ espressione2`	Eleva a potenza un numero in base al valore dell'esponente.	`5 ^ 3` che genera il risultato 125.
Mod	`espressione1 Mod espressione2`	Divide due valori interi e restituisce il resto.	`12 Mod 5` genera il risultato 2.

Uso delle classi

Per usare una classe è necessario, innanzitutto, creare una nuova istanza di tale classe.

Vedi anche: in questa parte, la sezione che descrive come dichiarare le classi.

La sintassi generale per creare un nuova istanza di classe è la seguente.

```
Dim istanzaClasse As New nomeClasse
```

Questa sintassi dimostra che la creazione di una nuova istanza di classe coinvolge le parole chiave `New`, `Dim` e `As` (di solito utilizzate per dichiarare le variabili). Dopo aver creato un'istanza di classe è necessario adottare per la stessa i metodi `Public` della classe di appartenenza. La sintassi generale per l'uso del metodo `Public` è la seguente.

```
istanzaClasse.nomeMetodo elencoValoriArgomento
```

Quello seguente è un esempio pratico dell'uso della classe `CMyInt`. Innanzitutto è necessario dichiarare la classe.

```
Private mintVal As Integer
Public Sub Store(X As Integer)
  mintVal = X
End Sub
Public Function Recall( ) As Integer
  Recall = mintVal
End Function
```

La classe dichiara l'elemento dati `mintVal` di tipo `Private Integer` e le routine `Store( )` e `Recall( )`. La procedura `Store( )` memorizza nell'elemento dati `mintVal` un nuovo valore (fornito dall'argomento `x`). La funzione `Recall( )` restituisce il valore dell'elemento dati `mintVal`.

Per dichiarare un'evenienza della classe `CMyInt` si dovrà ora scrivere

```
Dim objInt As New CMyInt
```

Quindi, per accedere all'istanza `objInt` si potranno scrivere enunciati simili ai seguenti.

```
objInt.Store(12)
Debug.Print objInt.Recall
```

Il primo enunciato richiama il metodo `Store` per memorizzare il numero 12 nell'istanza di classe `objInt`; il secondo enunciato richiama il valore nell'istanza di classe `objInt` servendosi del metodo `Recall( )`.

La classe potrà essere richiamata come argomento.

La sintassi generale per questa operazione è la seguente.

```
nomeArgomento As nomeClasse
```

Ecco un esempio pratico.

```
Sub CopyInt(SourceInt As CMyInt, TargetInt As CMyInt)
  TargetInt.Store(SourceInt.Recall)
End Sub
```

La procedura `CopyInt( )` copia il valore da un'istanza di classe `CMyInt` in un'altra istanza. La procedura ha gli argomenti `SourceInt` e `TargetInt` di tipo `CMyInt` il cui scopo è quello di specificare l'istanza di classe sorgente e quella di destinazione.

Uso delle costanti

Visual Basic consente di dichiarare nomi e valori fissi da associare a tali nomi. I programmatori chiamano questi nomi *costanti*. Le costanti possono essere dichiarate e utilizzate al posto di valori letterali, facilitando, così, la lettura e le operazioni di aggiornamento del codice. Per esempio, invece di avere un codice del tipo `If day = 7 Then`, a tale codice si potrà assegnare al valore 7 una costante (per esempio Domenica). A questo punto il frammento di codice precedente potrà essere scritto come `If day = Domenica Then`.

La sintassi generale per dichiarare una costante è la seguente.

```
[Public | Private] Const nomeCostante [As tipo] =
espressione
```

La parola chiave facoltativa `Public` viene inserita a livello di modulo per dichiarare che le costanti saranno disponibili per tutte le procedure di tutti i moduli. Non si ha la possibilità di dichiarare costanti di tipo `Public` all'interno delle procedure.

La parola chiave facoltativa `Private` viene inserita a livello del modulo per dichiarare le costanti che saranno disponibili solo all'interno del modulo nel quale è stata fatta la dichiarazione. Non si possono dichiarare costanti `Private` all'interno delle procedure.

L'argomento `nomeCostante` corrisponde al nome assegnato alla costante e deve seguire le convenzioni standard per l'assegnazio-

ne dei nomi alle variabili. L'argomento facoltativo `tipo` specifica il tipo di dati della costante (potrà essere `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String`, `Variant` o un qualsiasi tipo personalizzato). Se si omette l'argomento `tipo`, Visual Basic seleziona il tipo di dati che meglio si adatta al tipo dell'argomento `espressione`. Visual Basic richiede di utilizzare clausole `As tipo` per ogni costante che viene dichiarata. L'argomento `espressione` può essere di tipo letterale, un'altra costante o una qualsiasi combinazione che preveda operatori aritmetici o logici (con l'eccezione dell'operatore `Is`)

Nota. Nelle espressioni assegnate alle costanti non si possono usare variabili, funzioni personalizzate o funzioni Visual Basic predefinite (come, per esempio, `Chr( )`).

Viene ora proposto un esempio di costanti.

```
Const GIORNI_SETTIMANA = 7
Const MESI_ANNO As Byte = 12
Const COSTANTE_E As Double = 2.7183
Const Mois As String = "Mario Rossi"
```

Il primo esempio dichiara la costante `GIORNI_SETTIMANA` e associa alla stessa il valore 7. Poiché la dichiarazione non definisce alcun tipo di dati, Visual Basic assegnerà automaticamente alla costante il tipo di dati che ritiene più appropriato (per esempio `Byte` oppure `Integer`).

Il secondo esempio dichiara la costante `MESI_ANNO` alla quale viene associato il valore 12. In questo caso la dichiarazione imposta anche il tipo di dati `Byte`.

Il terzo esempio dichiara la costante `COSTANTE_E` e gli assegna il valore 2.7183. Viene inoltre definito il tipo di dati `Double`.

L'ultimo esempio, infine, dichiara la costante `Mois` e le associa la stringa di testo "Mario Rossi". La dichiarazione definisce inoltre il tipo di dati `String`.

L'uso delle costanti è relativamente semplice poiché richiede solo l'indicazione del nome della costante stessa. Si osservi l'esempio seguente.

```
Const GIORNI_SETTIMANA As Integer = 7
Dim NumeroSettimane As Integer
NumeroSettimane = 3
Debug.Print "Numero di giorni "; NumeroGiorni *
GIORNI_SETTIMANA
```

Questo esempio dichiara la costante `GIORNI_SETTIMANA` e le associa il valore 7 e il tipo di dati `Integer`. Viene inoltre dichiarata la

variabile NumeroSettimane, di tipo Integer e quindi le si assegna il valore 3. L'ultimo enunciato visualizza il numero dei giorni risultanti dalla moltiplicazione del valore nella variabile NumeroSettimane con quello della costante GIORNI_SETTIMANA. Il risultato di quest'ultimo enunciato visualizzerà, nella finestra Immediata, il risultato 21.

Uso degli operatori logici

Visual Basic viene fornito con una serie di operatori logici descritti nella tabella seguente. Questi operatori permettono di esaminare i valori logici di una o più espressioni di tipo booleano.

Operatore	*Sintassi*	*Commento*	*Esempio*
And	booleano1 And booleabo2	Esegue una congiunzione logica di due espressioni booleane (controlla se entrambe le espressioni sono vere).	(1 < 2) And (3 > 0) restituisce il risultato True poiché entrambe le espressioni sono vere.
Or	booleano1 Or booleano2	Esegue una separazione logica di due espressioni (controlla se almeno una delle due è vera).	(1 < 0) Or (3 > 0) restituisce il risultato True poiché la seconda espressione risulta vera.
Xor	booleano1 Xor booleano2	Esegue un'esclusione logica di due espressioni (controlla per verificare se solo una delle espressioni risulta vera).	(1 < 2) Xor (3 > 0) è False perché entrambe le espressioni sono vere.
Not	Not booleano	Esegue la negazione logica di una espressione	Not (1<0) è True poiché (1<0) è False.
Imp	booleano1 Imp booleano2	Esegue un'implicazione logica su due espressioni. Si comporta in modo diverso da And poiché False Imp True risulta vera (mentre False And True risulta falsa) e False Imp False risulta vera (mentre False And False risulta falsa).	(4 > 2) Imp (3 > 1) restituisce True poiché entrambe le espressioni sono vere.
Eqv	booleano1 Eqv booleano2	Esegue un'equivalenza logica di due espressioni. È diverso dall'operatore And in quanto False And False risulta falsa mentre False Eqv false risulta vera.	(4 > 2) Eqv (3 > 1) restituisce True poiché entrambe le espressioni sono vere.

Uso degli operatori relazionali

Gli operatori relazionali di Visual Basic, descritti nella tabella seguente, consentono di confrontare i valori di due espressioni.

Operatore	Sintassi	Commento	Esempio
>	espressione1 > espressione2	Restituisce True quando espressione1 è maggiore di espressione2. In caso contrario, False.	2 > 0 restituisce True 2 > 2 restituisce False 2 > 10 restituisce False
>=	espressione1 >= espressione2	Restituisce True quando espressione1 è maggiore o uguale a espressione2. In caso contrario, restituisce False.	2 >= 0 restituisce True 2 > = 2 restituisce True 2 > = 10 restituisce False
<	espressione1 < espressione2	Restituisce True quando espressione1 è minore di espressione2. In caso contrario restituisce False.	2 < 0 restituisce False 2 < 2 restituisce False 2 < 10 restituisce True
<=	espressione1 <= espressione2	Restituisce True se espressione1 è minore o uguale a espressione2. In caso contrario, restituisce False.	2 <= 0 restituisce False 2 <= 2 restituisce True 2 <= 10 restituisce True
=	espressione1 = espressione2	True quando espressione1 è uguale a espressione2. In caso contrario restituisce False.	2 = 0 restituisce False 2 = 2 restituisce True
< >	espressione1 < > espressione2	True quando espressione1 non è uguale a espressione2. In caso contrario restituisce False.	2 < > 0 restituisce True 2 < > 2 restituisce False
Is	oggetto1 Is oggetto2	Confronta due variabili di riferimento a oggetti. Restituisce True se la variabile fa riferimento allo stesso oggetto. In caso contrario restituisce False.	Set OggY = OggM Set OggTO = OggM IsSame = OggY Is OggTO Restituisce True poiché entrambe le variabili fanno riferimento allo stesso oggetto.
Like	stringa Like schema	Restituisce True se stringa soddisfa le specifiche definite da schema.	"WORD-EXE" Like "W*D.EXE" restituisce True.

Per maggiori informazioni sull'utilizzo degli operatori, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Uso dei tipi di dati personalizzati

Per usare un tipo di dati personalizzato è necessario dichiarare le variabili di tale tipo tramite un enunciato `Dim`.

Vedi anche: in questa parte, la sezione che descrive come creare tipi di dati personalizzati.

La sintassi generale per la dichiarazione di un tipo di dati personalizzato è la seguente.

```
Dim nomeVariabile As nomeTipo
```

Questa sintassi mostra che la procedura per la dichiarazione di un tipo di dati personalizzato è molto simile a quella utilizzata per dichiarare le variabili associate a tipi di dati predefiniti. Dopo aver dichiarato un tipo di dati personalizzato, si potrà accedere agli elementi dati servendosi dell'operatore "punto" (.).

La sintassi generale per accedere agli elementi dati di un tipo di dati personalizzato è la seguente.

```
TipoVarPersonale.elementodati
```

Quello che segue è un esempio di come sia possibile usare il tipo di dati `Punto`. Innanzitutto è necessario dichiarare il tipo di dati.

```
Type Punto
  XLoc As Integer
  YLoc As Integer
End Type
```

Il tipo `Punto` dichiara gli elementi dati `Xloc` e `YLoc` di tipo `Integer`.

Per dichiarare la variabile di tipo `Punto` si scriverà quanto segue.

```
Dim miaPosizione As Punto
```

A questo punto, per accedere agli elementi dati della variabile miaPosizione si potranno scrivere enunciati simili ai seguenti.

```
miaPosizione.Xloc = 12
miaPosizione.YLoc = 22
Debug.Print "("; miaPosizione.XLoc; ", ";
miaPosizione.YLoc; ")"
```

I primi due enunciati memorizzano i valori nell'elemento dati `XLoc` e `YLoc` della variabile `miaPosizione`.

L'ultimo enunciato visualizza, invece, i valori degli elementi dati accedendovi tramite l'enunciato `Print`.

Si ha anche la possibilità di utilizzare come argomento un tipo di dati personalizzato. La sintassi generale per questa operazione è la seguente.

```
nomeArgomento As nomeTipo
```

Ed ecco un esempio pratico.

```
Sub CopiaPunto(PuntoOrigine As Punto, PuntoDest As Punto)
  PuntoDest.XLoc = PuntoOrigine.XLoc
  PuntoDest.YLoc = PuntoOrigine.YLoc
End Sub
```

La procedura `CopiaPunto( )` copia il valore contenuto nell'argomento di tipo `Punto` in un altro argomento. La procedura prevede gli argomenti `PuntoOrigine` e `PuntoDest` di tipo `Punto` che specificano le coordinate di origine e quelle di destinazione.

Uso dei tipi enumerati personali

Per usare un tipo di dati enumerato personale è necessario prima dichiarare la variabile di tale tipo servendosi dell'enunciato `Dim`.

Vedi anche: in questa parte, la sezione che descrive come creare tipi di dati enumerati personalizzati.

La sintassi generale per dichiarare una variabile con tipo di dati enumerato personale è la seguente.

```
Dim nomevariabile As nomeTipo
```

Questa sintassi mostra che la procedura per la dichiarazione di una variabile di tipo enumerato personale è molto simile a quella utilizzata per le variabili che prevedono tipi di dati predefiniti. Dopo aver dichiarato un tipo di dati enumerato personale, si potrà accedere ai suoi valori servendosi dell'operatore "punto" (.). La sintassi generale per accedere agli elementi dati di un tipo di dati enumerato personale è la seguente.

```
VarTipoPersonale [= valoreEnumerazione]
```

Viene ora fornito un esempio di come sia possibile usare il tipo di dati enumerato personale `Weekend`. Innanzitutto si dichiara il tipo.

```
Enum Weekend
  Sabato = 7
  Domenica
End Enum
```

Il tipo `Weekend` dichiara i valori `Sabato` e `Domenica`.

Per dichiarare una variabile di tipo `Weekend` si potrà ora scrivere.

```
Dim mioGiorno As Weekend
```

A questo punto, per accedere ai valori della variabile `mioGiorno` si potranno scrivere enunciati simili ai seguenti.

```
mioGiorno = Domenica
If mioGiorno = Domenica Then
  Debug.Print "Domenica"
Else
  Debug.Print "Sabato"
End If
```

Il primo enunciato memorizza il valore enumerato `Domenica` all'interno della variabile `mioGiorno`. L'enunciato `If` confronta il valore contenuto nella variabile `mioGiorno` con `Domenica` e visualizza la stringa "Domenica" nel caso in cui i due valori siano coincidenti. In caso contrario, l'enunciato `If` permette di visualizzare la stringa "Sabato".

Un tipo di dati enumerato personale può essere usato anche come argomento.

La sintassi generale per questo tipo di argomento è la seguente.

```
nomeArgomento As nomeTipo
```

Ecco un esempio pratico.

```
Sub CopiaGiorno(GiornoOrigine As Weekend,
GiornoDest As Weekend)
  GiornoDest = GiornoOrigine
End Sub
```

La procedura `CopiaGiorno( )` copia il valore di tipo Weekend da un argomento di tipo Weekend all'altro.

Uso delle variabili

Le variabili sono destinate a memorizzare dati o riferimenti a oggetti. Anche se Visual Basic è in grado di creare variabili temporanee, è buona norma di programmazione dichiararle esplicitamente con un enunciato `Dim`. L'enunciato `Option Explicit` fa sì che le variabili debbano essere dichiarate e tale enunciato dovrà essere inserito nel modulo generale di dichiarazione del progetto e dovrebbe, inoltre, apparire come uno dei primi enunciati di un programma. L'enunciato `Option Explicit` indica al compilatore di emettere un messaggio di errore quando rileva la presenza di un testo che non è parte del linguaggio Visual Basic e quando tale testo non è stato precedentemente dichiarato. `Option Explicit` aiuta a evitare di commettere errori di digitazione consentendo quindi di correggere il codice.

La sintassi generale per dichiarare le variabili è la seguente.

```
Dim [WithEvents] nomeVar[([subscripts])] [As [New] tipo]
[, [WithEvents] nomeVar[([subscripts])] [As [New] tipo]]
...
```

La parola chiave facoltativa `WithEvents` specifica che `nomeVar` è una variabile oggetto utilizzato per rispondere a eventi gestiti da un oggetto ActiveX; la parola chiave `WithEvents` è valida solo nei moduli di classe. Sebbene con `WithEvents` si possano dichiarare tutte le variabili di cui si ha necessità, non si ha la possibilità di creare matrici. Inoltre non si può utilizzare la parola chiave `New` in congiunzione con la parola chiave `WithEvents`.

L'argomento nomeVar corrisponde al nome della variabile e deve assoggettarsi alle convenzioni standard per l'assegnamento dei nomi alle variabili. L'argomento facoltativo `subscripts` specifica le dimensioni di una variabile matrice e Visual Basic, in una variabile matrice multidimensionale, consente di dichiarare fino a 60 dimensioni. L'argomento `subscripts` si serve della seguente sintassi generale.

```
[min To] max [, [min To] max] ...
```

Quando si omette la parte `min`, la dichiarazione utilizza per tale parte il valore di default. L'enunciato `Option Base` controlla il valore minimo di default della matrice (che corrisponde, di solito, a 0 o a 1). La parola chiave `New` abilita la creazione implicita di un oggetto; l'argomento facoltativo tipo specifica il tipo di dati della variabile. Il valore di `tipo` può essere `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `String` (per una stringa di lunghezza variabile), `String * lunghezza` (per una stringa a lunghezza fissa), `Object`, `Variant` oppure un tipo di dati personalizzato o un tipo

oggetto. È necessario utilizzare la parola chiave `As` per ogni variabile che verrà dichiarata.

Qui di seguito vengono forniti alcuni esempi per la dichiarazione delle variabili.

```
Dim NumGiorni
NumGiorni = 7
Debug.Print NumGiorni

Dim NumMesi As Byte
NumMesi = 12
Debug.Print NumMesi

Dim VirgolaMobile As Double
VirgolaMobile = 2.7183
Debug.Print VirgolaMobile

Dim mioNome As String
mioNome = "Mario Rossi"
Debug.Print mioNome
```

Il primo esempio dichiara la variabile `NumGiorni` e poiché la dichiarazione non menziona alcun tipo di dati, Visual Basic ne selezionerà uno di tipo intero (come, per esempio, `Byte` oppure `Integer`). Il codice, quindi, assegna alla variabile il valore 7 e lo visualizza nella finestra Immediata.

Il secondo esempio dichiara la variabile `NumMesi`. La dichiarazione definisce anche il tipo di dati `Byte` e alla variabile viene assegnato il valore 12 che viene successivamente visualizzato nella finestra Immediata.

Il terzo esempio dichiara la variabile `VirgolaMobile` di tipo `Double`. Il codice assegna alla variabile il valore "2.7183" e lo visualizza nella finestra Immediata.

Il quarto esempio definisce la variabile `mioNome` di tipo `String`, le assegna il valore "Mario Rossi" che verrà quindi visualizzato nella finestra Immediata.

L'uso delle variabili è relativamente semplice poiché si richiede solo di indicare il nome di una costante, come nell'esempio seguente.

```
Const GIORNI_SETTIMANA As Integer = 7
Dim NumeroSettimane As Integer
NumeroSettimane = 3
Debug.Print "Numero di giorni "; NumeroGiorni *
GIORNI_SETTIMANA
```

Questo esempio dichiara la costante `GIORNI_SETTIMANA` e le associa il valore 7 e il tipo di dati `Integer`. Viene inoltre dichiarata la variabile `NumeroSettimane`, di tipo `Integer` e quindi le si assegna il valore 3. L'ultimo enunciato visualizza il numero dei giorni risultanti dalla moltiplicazione del valore nella variabile `NumeroSettimane` con quello della costante `GIORNI_SETTIMANA`. Il risultato di quest'ultimo enunciato visualizzerà, nella finestra Immediata, il risultato 21.

Le finestre di dialogo

Le finestre di dialogo hanno lo scopo di fungere da intermediario fra l'utente e l'applicazione. Indubbiamente questo tipo di conversazione tra utente e computer non è certamente degno di un premio Pulitzer ma lo scopo finale viene sempre e comunque raggiunto poiché tramite tali elementi si ha la possibilità di stampare, salvare e aprire documenti oppure avviare operazioni accessorie quali, per esempio, la definizione di tipo di carattere, delle dimensione o del colore del testo.

Argomenti trattati

- **Visualizzazione di messaggi e richieste di immissione**
- **Opzioni di stampa personalizzabili dall'utente**
- **File di guida relativi all'applicazione creata**
- **Finestre di dialogo per la modifica del colore e del tipo di carattere**
- **Finestre di dialogo per salvare e aprire i file dall'applicazione**

Finestra di dialogo "Colore"

La finestra di dialogo per la selezione di un colore è una di quelle standard in Windows 9x e consente all'utente di selezionare un colore prelevandolo da una palette (o di crearne uno personalizzato) da applicare al testo, agli sfondi e a molti altri elementi. Per accedere alla finestra di dialogo per la gestione del colore si deve utilizzare il metodo `ShowColor( )` del controllo `CommonDialog`. Per utilizzare la finestra di dialogo del colore si impostino le proprietà del controllo `CommonDialog` in modo che siano in grado di gestire tale finestra di dialogo.

Ricordare. La finestra di dialogo per la gestione del colore non è disponibile nella versione Standard di Visual Basic 6.

Nella tabella seguente vengono elencate le proprietà più importanti della finestra di dialogo per la gestione del colore.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commenti*
Color: `oggetto.`**`Color`** `[= numero]`	Restituisce o imposta il colore selezionato.	`dlgColor.Color = vbRed`	Assegna il colore rosso alla finestra di dialogo `dlg Color`.
Flags: `oggetto.`**`Flags`** `[= valore]`	Restituisce o imposta le opzioni per una finestra di dialogo per la gestione del colore (*vedi:* tabella successiva).	`shpBody.Fill Color = vbRed`	Imposta la proprietà `Fill Color` del controllo sul rosso.

Nella tabella seguente vengono invece elencati i valori accettabili della proprietà Flags della finestra di dialogo del colore.

Costante	*Valore*	*Descrizione*
`cdCClFullOpen`	**&H2**	Visualizza l'intera finestra di dialogo, compresa la sezione relativa alla definizione di colori personalizzati.
`cdlCCHelpButton`	**&H8**	Visualizza, all'interno della finestra di dialogo, il pulsante di comando ?.
`cdlCCPrevent FullOpen`	**&H4**	Disabilita il pulsante di comando per l'accesso all'area della finestra per la definizione dei colori personalizzati.
`cdlCCRGBInit`	**&H1**	Imposta il valore iniziale dei colori della finestra di dialogo.

La figura seguente è un esempio di programma che visualizza la finestra di dialogo per la gestione dei colori. Questo programma visualizza un modulo nel quale è contenuta un'etichetta e due pulsanti di comando: quello a sinistra richiama la finestra di dialogo colore per impostare il colore dello sfondo mentre quello di destra richiama la stessa finestra di dialogo per definire il colore del testo dell'etichetta.

Nella tabella seguente vengono elencate le proprietà del modulo e degli altri componenti utilizzati.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Demo Scelta Colori
Common Dialog	Name	dlgColor
Pulsante comando	Name Caption	cmdBackColor Colore &Sfondo
Pulsante comando	Name Caption	cmdForeColor Colore &Testo
Etichetta	Name	lblColor

Si propone ora il codice sorgente che genera la figura precedente.

```
Private Sub cmdBackColor_Click( )
  ' imposta la proprietà Flags
  dlgColor.Flags = cdlCCRGBInit
  ' imposta il colore della finestra di dialogo
  dlgColor.Color = lblColor.BackColor
  ' visualizza la finestra di dialogo del colore
```

```
  dlgColor.ShowColor
  lblColor.BackColor = dlgColor.Color
End Sub
Private Sub cmdForeColor_Click( )
  ' imposta la proprietà Flags
  dlgColor.Flags = cdlCCRGBInit
  ' imposta il colore della finestra di dialogo
  dlgColor.Color = lblColor.ForeColor
  ' visualizza la finestra di dialogo del colore
  dlgColor.ShowColor
  lblColor.ForeColor = dlgColor.Color
End Sub
Private Sub Form_Load( )
  lblColor.Caption = "Ciao Mondo!"
  lblColor.Font.Size = 24
End Sub
```

Questo codice contiene la procedura `Form_Load( )` che inizializza il form assegnando un testo alla proprietà `Caption` dell'etichetta e impostandone la dimensione del carattere a 24 punti.

La procedura `cmdBackColor_Click( )` risponde a un clic dell'utente sul pulsante di comando `Colore Sfondo` e gestisce l' evento eseguendo le seguenti operazioni.

- Imposta la proprietà `Flags` della finestra di dialogo del colore assegnandole come valore la costante `cdlCCRGBInit`.
- Imposta la proprietà `Color` della finestra di dialogo del colore assegnandole la proprietà `BackColor` del controllo Etichetta (questa operazione permette alla finestra di dialogo del colore di impostare il colore di sfondo dell'etichetta).
- Visualizza la finestra di dialogo del colore tramite il metodo `ShowColor( )`.
- Assegna il valore della proprietà `Color` (della finestra di dialogo del colore) alla proprietà `BackColor` del controllo Etichetta.

La procedura `cmdForeColor_Click( )` risponde a un clic dell'utente sul pulsante di comando `Colore Testo` e gestisce l'evento, eseguendo le operazioni seguenti.

- Imposta la proprietà `Flags` della finestra di dialogo del colore assegnandole come valore la costante `cdlCCRGBInit`.
- Imposta la proprietà `Color` della finestra di dialogo del colore assegnandole la proprietà `ForeColor` del controllo Etichetta (questa operazione permette alla finestra di dialogo del colore di impostare il colore del testo dell'etichetta).

- Visualizza la finestra di dialogo del colore tramite il metodo `ShowColor( )`.
- Assegna il valore della proprietà `Color` (della finestra di dialogo del colore) alla proprietà `ForeColor` del controllo Etichetta.

Finestra di dialogo "Carattere"

La finestra di dialogo "Carattere" (Font) è anch'essa una delle finestre standard di Windows 9x e consente all'utente di selezionare un tipo di carattere. Per visualizzare questa finestra di dialogo si deve utilizzare il metodo `ShowFont( )` del controllo `CommonDialog`. Per utilizzare la finestra di dialogo dei caratteri è necessario impostare correttamente le proprietà del controllo `CommonDialog`.

Ricordare. La finestra di dialogo Carattere non è disponibile nella versione Standard di Visual Basic 6.

Nella tabella seguente si riportano le proprietà più importanti della finestra di dialogo per la gestione dei caratteri.

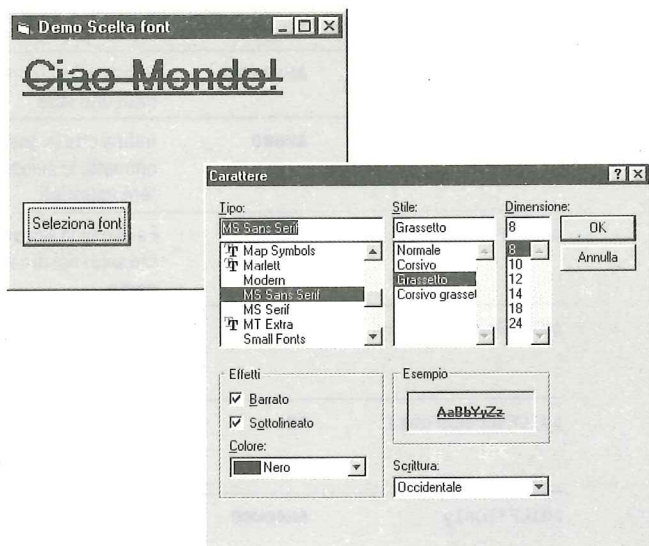
Proprietà	*Descrizione*
`Color`	Determina il colore selezionato. Per usare questa proprietà è necessario impostare la proprietà `Flags` su `cdlCFEffects`.
`FontBold`	Determina se è stato o meno selezionato lo stile **grassetto**.
`FontItalic`	Determina la selezione o meno dello stile *corsivo*.
`FontStrikeThrough`	Determina se è stato selezionato o meno lo stile ~~barrato~~. Per usare questa proprietà è necessario prima impostare la proprietà `Flags` su `cdlCFEffects`.
`FontUnderline`	Determina se è stato o meno selezionato lo stile <u>sottolineatura</u>. Per usare questa proprietà è necessario prima impostare la proprietà `Flags` su `cdl CFEffects`.
`FontName`	Determina il nome del carattere selezionato.
`FontSize`	Determina la dimensione del carattere selezionato.
`Min`	Determina la dimensione minima del carattere.
`Max`	Determina la dimensione massima del carattere.
`Flags`	Determina gli elementi per intervenire con più precisione nella finestra di dialogo.

La tabella seguente riporta, invece, i valori accettabili per la proprietà `Flags` della finestra di dialogo dei caratteri.

Costante	Valore	Descrizione
cdlCFANSIOnly	&H400	Specifica che la finestra di dialogo accetta solo la selezione di un carattere che utilizza il set di caratteri Windows. Con questa impostazione all'utente non sarà consentito di selezionare un tipo di carattere che contenga solo simboli.
cdlCFApply	&H200	Inserisce nella finestra di dialogo il pulsante di comando `Applica`.
cdlCFBoth	&H3	Fa sì che nella finestra di dialogo vengano visualizzati sia i caratteri della stampante e quelli video.
CdlCFEffects	&H100	Fa sì che la finestra di dialogo visualizzi anche l'attivazione degli effetti barrato, sottolineato e di colore.
cdlCFFixedPitchOnly	&H4000	Fa sì che la finestra di dialogo visualizzi solo tipi di carattere a spaziatura fissa.
cdlCFForceFontExist	&H10000	Fa sì che venga visualizzato un messaggio di errore quando l'utente cerca di selezionare un carattere o uno stile che non esiste.
cdlCFHelpButton	&H4	Inserisce nella finestra di dialogo il pulsante ?.
cdlCFLimitSize	&H2000	Specifica che la finestra di dialogo consente di utilizzare solo dimensioni di carattere contenute nell'intervallo specificato dalle proprietà Min e Max.
cdlCFNoFaceSel	&H80000	Indica che l'utente non ha selezionato un nome di carattere.
cdlCFNoSimulations	&H1000	Indica che la finestra di dialogo non consente l'uso delle simulazioni di tipi di carattere GDI (*Graphic Device Interface*).
cdlCFNoSizeSel	&H200000	Indica che l'utente non ha selezionato una dimensione di carattere.

Costante	*Valore*	*Descrizione*
`cdlCFNioStyleSel`	**&H100000**	Indica che l'utente non ha selezionato uno stile.
`cdlCFNoVectorFonts`	**&H800**	Indica che la finestra di dialogo non ammette la selezione di tipi di carattere vettoriali.
`cdlCFPrinterFonts`	**&H2**	Fa sì che la finestra di dialogo elenchi solo i tipi di carattere della stampante.
`cdlCFScalableOnly`	**&H20000**	Fa sì che la finestra di dialogo consenta di selezionare solo tipi di caratteri scalabili.
`cdlCFScreenFonts`	**&H1**	Fa sì che la finestra di dialogo elenchi soltanto i tipi di carattere video riconosciuti dal sistema.
`cdlCFTTOnly`	**&H40000**	Fa sì che la finestra di dialogo consenta solo la selezione di tipi di carattere TrueType.
`cdlCFWYSIWYG`	**&H8000**	Indica che la finestra di dialogo consente solo la selezione di tipi di carattere che sono disponibili sia per la stampante sia per il video. L'attivazione di questa impostazione prevede anche l'attivazione delle impostazioni cdlCFBoth e cdlCFScalableOnly.

Per maggiori informazioni sulla finestra di dialogo Carattere, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

La figura seguente è il risultato di un programma che utilizza la finestra di dialogo Carattere. Viene qui visualizzato un form contenente un'etichetta e un pulsante di comando per accedere alla finestra di dialogo Carattere dalla quale selezionare il tipo di carattere da applicare all'etichetta.

Nella tabella seguente vengono elencate le impostazioni delle proprietà degli oggetti del form.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Demo Scelta Font
Common Dialog	Name	dlfFont
Pulsante comando	Name Caption	cmdSelectFont Seleziona &Font
Etichetta	Name	lblFont

Ecco il codice sorgente che gestisce l'applicazione.

```
Private Sub cmdSelectFont_Click( )
  ' imposta la possibilità di selezionare un tipo
  ' e uno stile di carattere
  dlgFont.Flags = cdlCFScreenFonts Or cdlCFEffects
  ' imposta il colore del carattere
  dlgFont.Color = lblFont.Font
  ' imposta il nome del carattere
  dlgFont.FontName = lblFont.Font
  ' imposta la dimensione del carattere
  dlgFont.FontSize = lblFont.Font.Size
  ' imposta lo stile del carattere
```

```
  dlgFont.FontBold = lblFont.Font.Bold
  dlgFont.FontItalic = lblFont.Font.Italic
  dlgFont.FontUnderline = lblFont.Font.Underline
  dlgFont.FontStrikethru = lblFont.Font.Strikethrough
  ' visualizza la finestra di dialogo del carattere
  dlgFont.ShowFont
  ' aggiorna il tipo di carattere
  lblFont.FontName = dlgFont.FontName
  lblFont.FontSize = dlgFont.FontSize
  lblFont.ForeColor = dlgFont.Color
  lblFont.Font.Bold = dlgFont.FontBold
  lblFont.Font.Italic = dlgFont.FontItalic
  lblFint.Font.Underline = dlgFont.FontUnderline
  lblFont.Font.Strikethrough = dlgFint.FontStrikethru
End Sub
Private Sub Form_Load( )
  lblFont.Caption = "Ciao Mondo!"
End Sub
```

Questo codice contiene la procedura `Form_Load( )` che inizializza il modulo assegnando un testo alla proprietà `Caption` del controllo "Etichetta".

La procedura `cmdSelectFont_Click( )` gestisce l'evento clic dell'utente sul pulsante di comando eseguendo le operazioni seguenti.

- Imposta la proprietà `Flags` della finestra di dialogo Carattere al valore delle costanti `cdlCFScreenFonts` e `cdlCFEffects`.

- Imposta la proprietà `Color` della finestra di dialogo Carattere alla proprietà `ForeColor` del controllo "Etichetta". Questa operazione consente alla finestra di dialogo Carattere di avere lo stesso colore dell'etichetta quando il programma visualizzerà la finestra di dialogo.

- Imposta la proprietà `FontName` della finestra di dialogo Carattere alla proprietà `Font` del controllo Etichetta. Questa operazione consente alla finestra di dialogo Carattere di riportare lo stesso carattere dell'etichetta quando il programma aprirà la finestra di dialogo.

- Imposta la proprietà `FontSize` della finestra di dialogo Carattere a quella della proprietà `Font.Size` del controllo Etichetta in modo che, quando il programma aprirà la finestra di dialogo, questa riporti la stessa dimensione di carattere dell'etichetta.

- Imposta le proprietà di stile (grassetto, *corsivo*, <u>sottolineatura</u> e ~~barrato~~) della finestra di dialogo Carattere in base alle proprietà corrispondenti del controllo Etichetta.

Ciò fa sì che la finestra di dialogo Carattere, quando viene aperta dal programma, riporti lo stesso stile di carattere del controllo Etichetta.

- Visualizza la finestra di dialogo Carattere tramite il metodo `ShowFont( )`.
- Assegna i valori della finestra di dialogo relativi al tipo, dimensione, stile e colore del carattere alle corrispondenti proprietà del controllo Etichetta.

Finestra di dialogo "Guida"

La finestra di dialogo Guida fa parte di quelle standard di Windows 9x e consente all'utente di avviare il motore del sistema di guida di Windows e visualizzare il file impostato dalla proprietà `HlpFile`. Per visualizzare la finestra di dialogo Guida si deve utilizzare il metodo `ShowHelp( )` del controllo Common Dialog definendone le appropriate impostazioni.

Ricordare. La finestra di dialogo Guida non è disponibile nella versione Standard di Visual Basic 6.

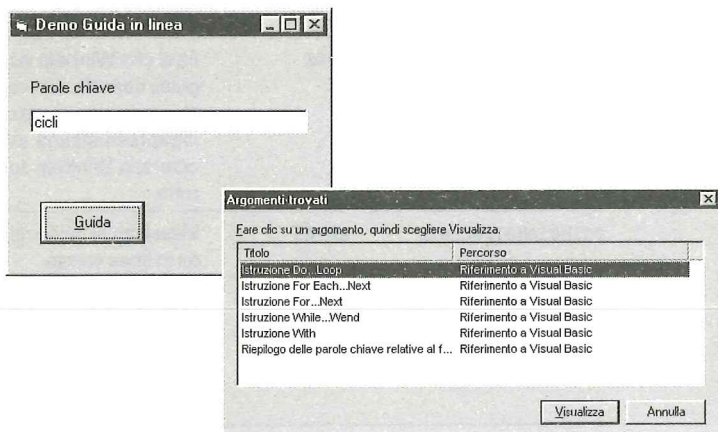
La tabella seguente riporta le proprietà più importanti della finestra di dialogo Guida.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
HelpCommand: `oggetto.`**`HelpCommand`** `[= valore]`	Restituisce o imposta il tipo di finestra guida in linea richiesta.	`dlgHelp.Help Command = cdlHelpkey`	Imposta il comando per l'accesso al sistema di guida per la ricerca dell'argomento specificato dalla proprietà Help Key.
HelpContext: `oggetto.`**`HelpContext`** `[= valore]`	Restituisce o imposta un numero di contesto associato a un oggetto. Utilizzato per fornire la guida contestuale dell'applicazione.	`dlgHelp.Help Context = 0`	Il valore 0 specifica l'assenza di contestualità. Un valore positivo specifica l'argomento contestuale.
HelpFile: `oggetto.`**`HelpFile`** `[= nomefile]`	Definisce il percorso e il nome di un file di guida Microsoft utilizzato dall'applicazione per visualizzare la guida o la documentazione in linea.	`dlgHelp.Help File = "vb6.hlp"`	Specifica che il file di guida corrisponde a VB6.HLP.
HelpKey: `oggetto.`**`HelpKey`** `[= stringa]`	Restituisce o imposta la parola chiave che identifica l'argomento della guida desiderato.	`dlfHelp.Help Key = "cicli"`	Specifica che cicli è uno degli argomenti per i quali si richiede aiuto.

La tabella seguente riporta, invece, i valori ammessi dalla proprietà HelpCommand della finestra di dialogo Guida.

Costante	*Valore*	*Descrizione*
cdlHelpCommand	**&H102&**	Esegue una macro della Guida in linea.
cdlHelpCintents	**&H3&**	Visualizza il sommario della guida contestuale definito dall'opzione Contents della sezione [Option] del file .HPJ.
cdlHelpContext	**&H15&**	Visualizza la guida relativa a un contesto particolare. Quando si utilizza questa impostazione è necessario anche specificare un contesto tramite la proprietà HelpContext.
cdlHelpContextPopup	**&H8&**	Visualizza in una finestra a comparsa (popup) un argomento di guida particolare, definito nella sezione [MAP] del file .HPJ.
cdlHelpForceFile	**&H9&**	Fa sì che WinHelp visualizzi il file di guida corretto. Nel caso in cui questo risulti già visualizzato, non verrà intrapresa alcuna azione. In caso contrario WinHelp aprirà il file corretto.
cdlHelpHelpOnHelp	**&H14&**	Visualizza la guida relativa alla Guida in linea stessa.
cdlHelpIndex	**&H3&**	Visualizza l'indice del file della guida specificato. Un'applicazione dovrebbe usare questo valore solo per un file di guida contenente un indice singolo.
cdlHelpKey	**&H101&**	Visualizza la guida di una parola chiave particolare. Quando si utilizza questa opzione, sarà necessario specificare anche una parola chiave servendosi della proprietà HelpKey.
cdlHelpPartialKey	**&H105&**	Visualizza l'argomento reperito nell'elenco delle parole chiave che coincide con i caratteri di ricerca digitati.

Costante	*Valore*	*Descrizione*
`cdlHelpQuit`	&H2&	Notifica all'applicazione di guida che il file di guida non è più in uso.
`cdlHelpSetContext`	&H5&	Determina quale argomento contestuale verrà visualizzato quando l'utente premerà il tasto F1 (Guida).
`cdlHelpSetindex`	&H5&	Imposta il contesto specificato dalla proprietà HelpContext come indice corrente del file di guida definito dalla proprietà HelpFile. Questo indice rimane attivo fino a quando l'utente accede a un diverso file di guida. Utilizzare questo valore solo per i file di guida contenenti più di un indice.

La figura seguente è il risultato di un programma che visualizza, insieme al form, una finestra di dialogo Guida.

Nella tabella seguente vengono riportate le impostazioni degli oggetti definiti.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Demo Guida in linea
Common Dialog	Name HelpFile HelpContext	dlgHelp vb6.hlp 3
Pulsante di comando	Name Caption	cmdHelp &Guida
Etichetta	Name Caption	lblKey Parole chiave
Casella di testo	Name	txtKey

Ecco il codice sorgente dell'applicazione.

```
Private Sub cmdHelp_Click( )
  dlgHelp-HelpCommand = cdlHelpKey
  dlgHelp.HelpKey = txtKey.Text
  dlgHelp.ShowHelp
End Sub
Private Sub Form_Load( )
  txtKey.Text = "cicli"
End Sub
```

La procedura `Form_load( )` inizializza il modulo assegnandogli la stringa letterale "cicli" per definire l'argomento di guida iniziale da visualizzare nella casella di immissione.

La procedura `cmdHelp_Click( )` gestisce l'evento clic sul pulsante di comando `Guida` eseguendo le operazioni seguenti.

- Assegna alla proprietà `HelpCommand` della finestra di dialogo Guida il valore della costante `cdlHelpKey`. Il valore assegnato fa sì che la finestra di dialogo richiami la guida relativa all'argomento specificato dalla proprietà `HelpKey`.
- Copia la proprietà `Text` della casella di immissione nella proprietà `HelpKey` della finestra di dialogo Guida.
- Servendosi del metodo `ShowHelp( )`, richiama la finestra di dialogo Guida.

Finestra di dialogo "Immissione" (InputBox)

In Visual Basic esiste la funzione `InputBox( )` il cui scopo è quello di aprire una finestra di dialogo per immettere dati all'interno dei

programmi Visual Basic. Una finestra di dialogo di questo tipo visualizza una richiesta che rimane in attesa di immissione da parte dell'utente o di un'operazione di clic, generando quindi la memorizzazione della stringa contenuta nella casella di immissione della finestra di dialogo InputBox. Le finestre di dialogo di questo tipo possono essere usate per richiedere all'utente di eseguire un'operazione particolare prima che il programma proceda (per esempio, in una finestra InputBox che richiede conferma di un'operazione, potrebbe essere necessario fare clic sul pulsante di comando Ok oppure su quello Annulla).

La sintassi generale della funzione `InputBox( )` è la seguente.

```
InputBox(messaggio[, titolo] [, default] [, poxX] [, posY]
[, fileHelp, contesto])
```

L'argomento `messaggio` rappresenta un'espressione stringa che viene visualizzata come messaggio all'interno della finestra di dialogo InputBox. Il valore di questo argomento può essere composto da un massimo di 1.024 caratteri (il valore esatto dipende dalla dimensione del carattere utilizzato). Se il valore dell'argomento `messaggio` contiene più righe, queste potranno essere separate tramite un carattere di ritorno carrello (`Chr(13)`), da un carattere di avanzamento riga (`Chr(10)`) o da una combinazione dei due caratteri (`Chr(13) + Chr(10)`).

L'argomento `titolo` corrisponde a un'espressione stringa che specifica il titolo che apparirà nella barra corrispondente della finestra di dialogo InputBox. Quando non si inserisce l'argomento `titolo`, la finestra di dialogo InputBox visualizzerà, nella propria barra del titolo, il nome dell'applicazione. L'argomento facoltativo `default` è una espressione stringa che viene visualizzata nella casella di immissione quando viene aperta per la prima volta la finestra di dialogo InputBox. Rappresenta la risposta di default che il programma si aspetta di ottenere dall'utente se questi non inserisce nulla nella casella di immissione. Quando si omette di indicare un valore per l'argomento `default`, la casella di immissione, quando viene aperta la finestra di dialogo InputBox, apparirà vuota.

L'argomento facoltativo `posX` corrisponde a un'espressione numerica che specifica la distanza orizzontale (espressa in twips) del margine sinistro della finestra di dialogo a partire dal margine sinistro dello schermo. Il twip (vedi glossario) rappresenta un'unità di misura indipendente da qualunque periferica. Quando si omette di indicare un valore per tale argomento, la finestra di dialogo Input Box apparirà orizzontalmente centrata. L'argomento facoltativo `posY`, invece, corrisponde a una espressione numerica che definisce la distanza verticale (espressa in twips) del margine superiore della finestra di dialogo rispetto al margine superiore dello schermo.

Anche in questo caso, se si omette di assegnare un valore all'argomento `posY`, Visual Basic posizionerà la finestra centrata in verticale (ovvero il bordo superiore apparirà a circa 1/3 dell'altezza dello schermo).

L'argomento facoltativo `fileHelp` è un'espressione stringa che specifica il file di guida contenente l'aiuto contestuale per la finestra di dialogo. Quando si assegna un valore a questo argomento è necessario fornire anche un valore per l'argomento `contesto`. Quest'ultimo argomento (facoltativo) corrisponde a un'espressione numerica che rappresenta il numero del sistema di guida contestuale assegnato all'argomento stesso dall'autore del file di guida richiamato. Quando si fornisce un valore per l'argomento `contesto`, è necessario anche assegnare un valore all'argomento `fileHelp`.

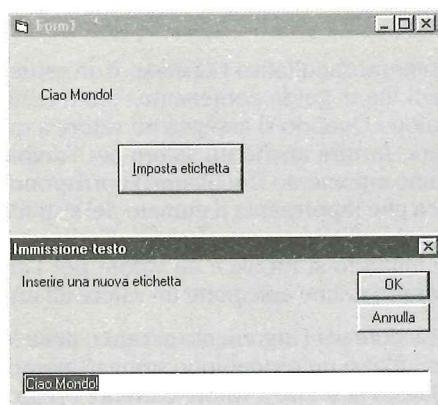
Come valore per l'argomento `default` della finestra di dialogo InputBox, si utilizzi un'espressione stringa (memorizzata in una variabile). Questo fa sì che il valore corrente venga considerato come default, evitando, quindi di dover utilizzare un valore fisso.

Viene ora proposto un esempio su come usare una finestra di dialogo InputBox. L'esempio è basato su un pulsante di comando sul quale si deve fare clic per visualizzare la finestra di dialogo stessa. Il testo che l'utente digiterà viene automaticamente assegnato alla proprietà `Caption` di un controllo etichetta (al quale è stato assegnato il nome `lblText`).

```
Private Sub cmdSetLabel_Click( )
  lblText.Caption = InputBox("Inserire una nuova
etichetta", "Immissione testo", lblText.Caption)
End Sub
```

La procedura imposta il risultato della funzione `InputBox( )` come nuovo valore per la proprietà `Caption` del controllo Etichetta. La chiamata della funzione `InputBox( )` si serve dei seguenti valori di argomento.

- Il valore dell'argomento `prompt` corrisponde alla stringa letterale `"Inserire una nuova etichetta"` e, pertanto, la finestra di dialogo visualizzerà tale valore.
- Il valore dell'argomento titolo corrisponde alla stringa letterale `"Immissione testo"` che viene visualizzata nella finestra di dialogo per comunicare all'utente che deve scrivere una stringa letterale (e non numerica).
- Il valore dell'argomento `default` corrisponde alla proprietà `Caption` del controllo Etichetta. Pertanto il valore di default diventa il contenuto dell'etichetta stessa.

La chiamata della funzione utilizza i valori di default anche per gli altri argomenti e la figura seguente è il risultato ottenuto.

Finestra di dialogo "Messaggio" (MsgBox)

Tramite la funzione `MsgBox( )` Visual Basic consente di visualizzare una finestra di dialogo MsgBox. Questa finestra di dialogo potrà essere utilizzata per richiedere semplicemente all'utente di fare clic su un pulsante di comando di conferma, di rinuncia o di annullamento di una operazione

La finestra di dialogo MsgBox richiede esplicitamente la pressione di un pulsante e il risultato ottenuto è un valore intero che rappresenta il pulsante sul quale l'utente ha fatto clic.

La sintassi generale della funzione `MsgBox( )` è la seguente.

```
MsgBox(messaggio[, pulsanti] [' titolo] [, fileHelp,
contesto])
```

L'argomento `messaggio` rappresenta un'espressione stringa che viene visualizzata come messaggio all'interno della finestra di dialogo Immissione. Il valore di questo argomento può essere composto da un massimo di 1.024 caratteri (il valore esatto dipende dalla dimensione del carattere utilizzato). Se il valore dell'argomento `messaggio` contiene più righe, queste potranno essere separate tramite un carattere di ritorno carrello (`Chr(13)`), da un carattere di avanzamento riga (`Chr(10)`) o da una combinazione dei due caratteri (`Chr(13) + Chr(10)`).

L'argomento facoltativo `pulsanti` corrisponde, invece, a un'espressione numerica che è la somma dei valori che specificano gli elementi seguenti.

- Numero e tipo dei pulsanti da visualizzare.
- Stile di icona da utilizzare.
- Identità del pulsante di default.
- Modalità della finestra di dialogo MsgBox . Le finestre di dialogo modali richiedono che l'utente effetui una scelta prima che l'applicazione possa procedere. Le finestre di dialogo non modali, al contrario, non richiedono tale operazione consentendo comunque al programma di procedere con la propria esecuzione.

Quando non si assegna un valore all'argomento `pulsanti`, a esso viene assegnato il valore di default 0. L'argomento `titolo` è un'espressione stringa che specifica il nome che dovrà apparire nella barra del titolo della finestra MsgBox. Se non si specifica un valore per tale argomento, Visual Basic assegna automaticamente alla finestra il nome dell'applicazione.

L'argomento facoltativo `fileHelp` è un'espressione stringa che specifica il file di guida che contiene l'aiuto contestuale per la finestra di dialogo. Quando si assegna un valore a questo argomento è necessario fornire anche un valore per l'argomento `contesto`. Quest'ultimo argomento (facoltativo) corrisponde a un'espressione numerica che rappresenta il numero del sistema di guida contestuale assegnato all'argomento stesso dall'autore del file di guida richiamato. Quando si fornisce un valore per l'argomento `contesto`, è necessario anche assegnare un valore all'argomento `fileHelp`.

Nella tabella seguente vengono schematizzati i valori ammissibili per l'argomento `pulsanti`.

Costante	*Valore*	*Descrizione*	*Commenti*
`vbOKOnly`	0	Visualizza solo il pulsante di comando OK.	Definisce il numero e il tipo di pulsanti.
`vbOKCancel`	1	Visualizza i pulsanti OK e Annulla.	Definisce il numero e il tipo di pulsanti.
`vbAbortRetryIgnore`	2	Visualizza i pulsanti di comando Termina, Riprova e Ignora.	Definisce il numero e il tipo di pulsanti.
`vbYesNoCancel`	3	Visualizza i pulsanti Sì, No e Annulla.	Definisce il numero e il tipo di pulsanti.
`vbYesNo`	4	Visualizza i pulsanti di comando Sì e No.	Definisce il tipo e il numero dei pulsanti.

continua

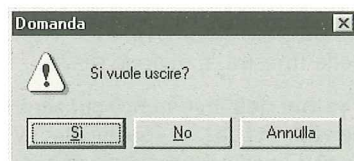
Costante	*Valore*	*Descrizione*	*Commenti*
`vbRetryCancel`	5	Visualizza i pulsante Riprova e Annulla.	Definisce il numero e il tipo di pulsanti.
`vbCritical`	16	Visualizza l'icona di Messaggio critico.	Definisce uno stile di icona.
`vbQuestion`	32	Visualizza un'icona di attenzione (punto interrogativo).	Definisce uno stile di icona.
`vbExclamation`	48	Visualizza un'icona di richiesta di avviso (punto esclamativo).	Definisce uno stile di icona.
`vbInformation`	64	Visualizza un'icona di messaggio di avviso (contenente una "i").	Definisce uno stile di icona.
`vbDefaultButton1`	0	Il primo pulsante viene considerato come quello predefinito.	Definisce un pulsante di default.
`vbDefaultButton2`	256	Il secondo pulsante viene considerato predefinito.	Definisce un pulsante di default.
`vbDefaultButton3`	512	Imposta il terzo pulsante come attivo per default.	Definisce un pulsante di default.
`vbDefaultButton4`	768	Rende predefinito il quarto pulsante.	Definisce un pulsante di default.
`vbApplicationModal`	0	Applicazione modale: l'utente deve rispondere alla richiesta della finestra di dialogo prima di poter procedere con l'applicazione.	Definisce una modalità.
`vbSystemModal`	4096	Sistema modale: tutte le applicazioni vengono sospese fintanto che l'utente non risponderà alla richiesta della finestra.	Definisce una modalità.
`vbMsgBoxHelpButton`	16384	Inserisce il pulsante per richiamare la guida.	

continua

Costante	*Valore*	*Descrizione*	*Commenti*
`vbMsgBoxSetForeground`	65536	Imposta la finestra di dialogo come elemento che deve apparire sempre in primo piano.	Definisce una modalità.
`vbMsgBoxRight`	524288	Il testo viene allineato a destra.	Definisce una modalità.
`vbMsgBoxRtlReading`	1048576	Specifica che il testo dovrebbe apparire disposto da destra a sinistra per consentire una lettura sui sistemi ebraici o arabi.	Definisce una modalità.

La tabella seguente riporta i valori restituiti dalla funzione `MsgBox( )`.

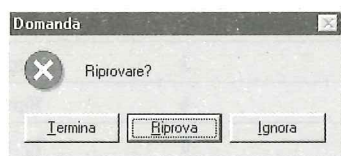
Costante	*Valore*	*Descrizione*
`vbOK`	1	OK
`vbCancel`	2	Annulla
`vbAbort`	3	Interrompi
`vbRetry`	4	Riprova
`vbIgnore`	5	Ignora
`vbYes`	6	Sì
`vbNo`	7	No

Vengono ora proposti alcuni brevi esempi dell'uso della funzione `MsgBox( )`. Il primo visualizza una finestra di dialogo "Sì/No/Annulla" mostrata in figura.

Il codice che genera questa finestra di dialogo è il seguente.

```
msgRisultato = MsgBox("Si vuole uscire?", vbYesNoCancel +
vbDefaultButton1 + vbExclamation)
```

In questo esempio viene richiamata la funzione `MsgBox( )` associata ai seguenti valori di argomento.

- Il valore per l'argomento `messaggio` corrisponde alla stringa `"Si vuole uscire?"` che viene visualizzata all'interno della finestra di dialogo.
- Il valore per l'argomento pulsanti corrisponde all'espressione `vbYesNoCancel + vbDefaultButton1 + vbExclamation`. Questa espressione fa sì che nella finestra di dialogo vengano visualizzati gli elementi seguenti.
    - Pulsanti "Sì", "No" e "Annulla".
    - Attivazione per default del primo pulsante.
    - Visualizzazione dell'icona contenente il punto esclamativo.
- Il valore per l'argomento `titolo` corrisponde alla stringa letterale `"Domanda"` che viene riportata a livello della barra del titolo della finestra.

L'esempio successivo visualizza una finestra di dialogo di tipo "Termina/Riprova/Ignora", visualizzata in figura.

Il codice che genera questa finestra di dialogo è il seguente.

```
msgRisultato = MsgBox("Riprovare?", vbCritical +
vbAbortRetryIgnore + vbDefaultButton2, "Errore!")
```

In questo esempio viene richiamata la funzione `MsgBox( )` associata ai seguenti valori di argomento.

- Il valore per l'argomento `messaggio` corrisponde alla stringa letterale `"Riprovare?"` che viene riportata all'interno della finestra di dialogo stessa.
- Il valore dell'argomento `pulsanti` è definito dall'espressione `vbCritical + vbAbortRetryIgnore + vbDefaultButton2`. Questa espressione genera la visualizzazione dei seguenti elementi.
    - Pulsanti "Termina", "Riprova" e "Ignora".
    - Attivazione per default del secondo pulsante.
    - Visualizzazione dell'icona di errore critico (una x).

✦ Il valore per l'argomento `titolo` corrisponde alla stringa letterale `"Errore!"` che viene riportata a livello della barra del titolo della finestra stessa.

Finestra di dialogo "Apri/Salva con nome"

Le finestre di dialogo Apri (Open) e Salva con nome (Save As) sono conformi alle due corrispondenti versioni di finestre standard di Windows 9x. Per visualizzare queste finestre di dialogo si devono utilizzare, rispettivamente, i metodi `ShowOpen( )` e `ShowSave( )` del controllo `CommonDialog`. Entrambe le finestre di dialogo consentono di specificare l'unità disco, la cartella, l'estensione e il nome del file. La finestra di dialogo Salva con nome è pressoché identica alla finestra di dialogo `Apri` con la sola eccezione dell'etichetta che appare nella barra del titolo della finestra stessa. In fase di esecuzione dell'applicazione, quando l'utente seleziona un file e attiva la finestra di dialogo, la proprietà `Filename` memorizza il nome del file selezionato. Si ha anche la possibilità di impostare la proprietà `Filter` in modo tale che nella finestra di dialogo vengano visualizzati solo i nomi di alcuni tipi di file (per esempio quelli di testo o altri documenti). La proprietà `Flags` influisce su alcuni elementi della finestra di dialogo e fa sì che l'utente venga avvisato quando vengono avviate alcune operazioni come, per esempio, la sovrascrittura di un file.

Ricordare. Le finestre di dialogo Apri e Salva con nome non sono disponibili nell'edizione Standard di Visual Basic.

Per maggiori informazioni sulle finestre di dialogo Apri e Salva con nome, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Nella tabella seguente vengono riportate le proprietà più importanti della finestra di dialogo Apri.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
DefaultExt: `oggetto.DefaultExt [= stringa]`	Restituisce o imposta l'estensione di default per i nomi di file riportati all'interno della finestra di dialogo.	`dlgOpen.Default.Ext = ".txt"`	Assegna alla finestra di dialogo `dlgOpen` l'estensione di default .txt.
FileName: `oggetto.Filename [= stringa]`	Restituisce o imposta il percorso e il nome del file selezionato.	`dlgOpen.FileName = "leggimi.txt"`	Assegna alla proprietà `FileName` il valore "leggimi.txt".

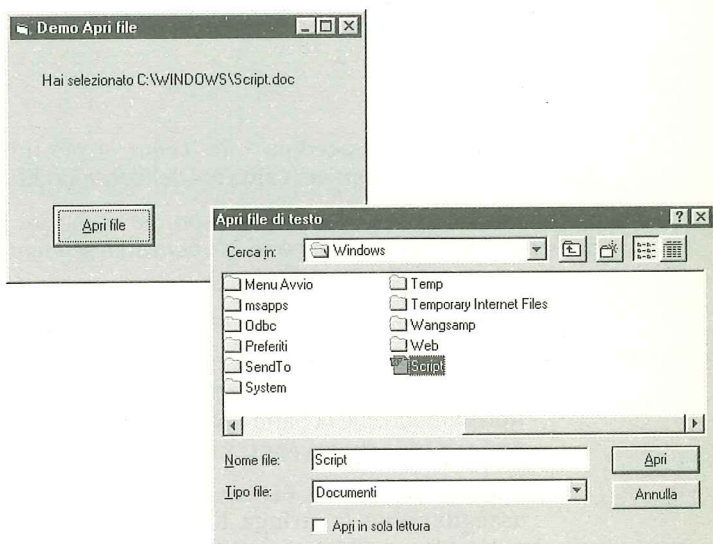
continua

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
FileTitle: `oggetto.`**`FileTitle`**	Restituisce il nome (senza indicarne il percorso) del file da aprire o da salvare.	`mioFile = dlgOpen. FileTitle`	Memorizza nella variabile mioFile il nome del file selezionato.
Filter: `oggetto.`**`Filter`** `[= stringa Filtro]`	Restituisce o imposta i filtri che dovranno apparire nella casella di elenco "Tipo file" della finestra di dialogo.	`dlgOpen. Filter = "Testo (*.txt)¦*.txt ¦Immagini (*.bmp;*.ico)¦ *.bmp;*.ico"`	Imposta i filtri per accedere ai file di solo testo (*.TXT) o a quelli di tipo grafico (*.BMP e *.ICO).
FilterIndex: `oggetto.`**`FilterIndex`** `[= numero]`	Restituisce o imposta un filtro di default per una finestra di dialogo Apri o Salva con nome. L'indice relativo al primo filtro corrisponde a 1.	`dlgOpen.Filter Index = 2`	Attiva per default la selezione del secondo filtro.
DialogTitle: `oggetto.`**`DialogTitle`** `[= stringa]`	Restituisce o imposta la stringa visualizzata nella barra del titolo della finestra di dialogo.	`dlgOpen. DialogTitle = "Seleziona un file di testo"`	Applica alla barra del titolo della finestra di dialogo la stringa indicata.
Flags: `oggetto.`**`Flags`** `[= valore]`	Restituisce o imposta le opzioni per una finestra di dialogo Apri o Salva con nome (si veda la tabella successiva).	`dlgOpen.Flags = cdlOFNLong Names`	Imposta la proprietà Flags in modo che la finestra di dialogo sia in grado di riconoscere il nome lungo dei file.
InitDir: `oggetto.`**`InitDir`** `[= stringa]`	Restituisce o imposta la cartella iniziale.	`dlgOPpen.InitDir = "\"`	Imposta la cartella radice come elemento iniziale.
MaxFileSize: `oggetto.`**`MaxFileSize`** `[= valore]`	Restituisce o imposta la dimensione massima dei nomi di file aperti servendosi del controllo Common Dialog. L'argomento valore corrisponde a un numero intero che specifica la dimensione massima (espressa in byte) del nome di file. L'intervallo ammissibile per questa proprietà va da 1K a 32 K e la dimensione assunta per default è pari a 256 byte.	`dlgOpen. MaxFileSize = 512`	Imposta la dimensione massima del file a 512 byte.

La tabella seguente riporta i valori ammissibili per la proprietà `Flags` delle finestre di dialogo Apri e Salva con nome.

Costante	*Valore*	*Descrizione*
cdlOFNAllowMultiselect	**&H200**	Fa sì che nell'area dei nomi dei file della finestra di dialogo venga attivata la possibilità di selezione multipla di file. L'utente può selezionare più di un file premendo e mantenendo premuto il tasto Maiusc e utilizzando quindi i tasti ↑ o ↓ per selezionare i file desiderati. L'utente potrà anche selezionare più file non contigui premendo e mantenendo premuto il tasto Ctrl e facendo clic sui singoli file che desidera selezionare. In questo caso, la proprietà `FileName` restituisce una stringa contenente i nomi dei file selezionati, separati l'uno dall'altro da uno spazio.
cdlOFNCreatePrompt	**&H2000**	Fa sì che la finestra di dialogo richieda all'utente di creare un file che non esiste ancora. Questa impostazione definisce automaticamente i marcatori `cdlOFNPathMustExist` e `cdlOFNFileMustExist`.
cdlOFNExplorer	**&H80000**	Si serve di una finestra di dialogo simile a quella di Explorer. Si tratta di un'impostazione operativa solo con Windows 9x o Windows NT 4.0.
cdlOFNExtensionDifferent	**&H400**	Indica che l'estensione del nome del file restituito è diversa da quella specificata dalla proprietà `DefaultExt`. Questa impostazione non è possibile se la proprietà `DefaultExt` è impostata su `Null`,se l'estensione è corrispondente oppure se il file non prevede estensione. Il valore può essere verificato chiudendo la finestra di dialogo.
cdlOFNFileMustExist	**&H1000**	Fa sì che l'utente possa inserire nell'area di visualizzazione dell'elenco dei file solo nomi già esistenti. Con l'attivazione di questa impostazione, se l'utente inserisce un nome di file non valido, viene visualizzato un messaggio di avvertimento. Questa impostazione attiva automaticamente anche le impostazioni per `cdlOFN PathMustExist`.
cdlOFNHelpButton	**&H10**	Inserisce nella finestra di dialogo il pulsante di comando per accedere alla guida.

Costante	Valore	Descrizione
cdlOFNHideReadOnly	&H4	Nasconde la casella di controllo per l'assegnazione dell'opzione di sola lettura dei file.
cdlOFNLongNames	&H200000	Consente l'uso di nomi lunghi di file.
cdlOFNNoChangeDir	&H8	Fa sì che l'indicazione della cartella corrente venga riportata a quella originaria quando la finestra di dialogo è stata aperta.
cdlOFNNoDereferenceLinks	&H100000	Fa sì che i collegamenti non vengano "annullati". Per default questa impostazione attiva una "dereferenziazione" dei collegamenti, cioè determina la sostituzione di un collegamento con l'oggetto cui si riferisce.
cdlOFNNoLongNames	&H40000	Non consente l'uso di nomi lunghi di file.
cdlOFNNoReadOnlyReturn	&H8000	Fa sì che al file restituito non venga applicato l'attributo di sola lettura né inserito in una cartella di tipo protetto.
cdlOFNNoValidate	&H100	Fa si che la finestra di dialogo comune accetti anche nomi di file con caratteri non validi.
cdlOFNOverritePrompt	&H2	Fa sì che la finestra di dialogo Salva con nome visualizzi un messaggio di avvertimento prima di sovrascrivere un file già esistente.
cdlOFNPathNameMustExist	&H800	Fa sì che l'utente possa inserire solo percorsi di memorizzazione esistenti. Quando questa impostazione è attiva e l'utente scrive un percorso di memorizzazione non esistente, la finestra di dialogo visualizza un messaggio di avvertimento.
cdlOFNreadOnly	&H1	Fa sì che la casella di controllo relativa all'opzione per l'impostazione dell'attributo di sola lettura per i file sia inizialmente attiva. L'attivazione di questa impostazione consente anche di controllare lo stato della casella di controllo quando la finestra di dialogo viene chiusa.
cdlOFNShareAware	&H4000	Fa sì che vengano ignorati gli errori di violazione della condivisione.

La figura seguente è un esempio di applicazione che visualizza una finestra di dialogo Apri.

La tabella seguente riporta le impostazioni delle proprietà degli oggetti presenti nella figura precedente.

Oggetto	Proprietà	Impostazione
Form	Name Caption	Form1 (default) Demo Apri
Common Dialog	Name	dlgOpen
Pulsante comando	Name Caption	cmdOpen Apri file
Etichetta	Name	lblFile

Si propone ora il codice sorgente per la generazione della finestra di dialogo Apri.

```
Private Sub cmdOpen_Click( )
  dlgOpen.InitDir = "\"
  dlgOpen.DialogTitle = "Apri file di testo"
  dlgOpen.Filter = "File di testo¦*.txt¦" + "Documenti
    ¦*.doc¦¦"
  dlgOpen.FilterIndex = 2
  ' visualizza la finestra di dialogo Open
  dlg.Open.ShowOpen
  lblFile.Caption = "Hai selezionato " + dlgOpen.FileName
```

```
End Sub
Private Sub Form_Load( )
  lblFile.Caption = ""
End Sub
```

Il codice utilizza la procedura `Form_Load( )` per inizializzare il form svuotando la proprietà `Caption` del controllo Etichetta.

La procedura `cmdOpen_Click( )` risponde al clic sul pulsante di comando `Apri File` eseguendo le operazioni seguenti.

- Imposta la proprietà `InitDir` della finestra di dialogo Apri a "\" in modo da definire la cartella radice come elemento iniziale.
- Imposta la proprietà `DialogTitle` della finestra di dialogo Apri in modo che la barra del titolo della finestra di dialogo Apri riporti la stringa `Apri file di testo`.
- Imposta la proprietà `Filter` della finestra di dialogo Apri assegnandole la stringa letterale `file di testo¦*.txt¦Document¦*.doc¦¦"`. Questa stringa contiene due filtri: uno per i file di solo testo (*.TXT) e il secondo per i file documento (*.DOC).
- Imposta a 2 la proprietà `FilterIndex` della finestra di dialogo Apri in modo che all'apertura della finestra risulti selezionato il filtro relativo il file documento (*.DOC).
- Apre la finestra di dialogo Apri con il metodo `ShowOpen( )`.
- Imposta la proprietà `Caption` dell'etichetta al valore della proprietà `FileName` della finestra di dialogo Apri. Questa operazione riflette visivamente il nome del file selezionato dall'utente.

Finestra di dialogo "Stampa"

La finestra di dialogo Stampa fa parte delle finestre standard di Windows 9x e consente all'utente di specificare un intervallo di pagine da stampare, la qualità di stampa, il numero delle copie, la loro eventuale fascicolazione e la creazione di un file di stampa su disco.

Questa finestra di dialogo contiene anche le informazioni relative alla o alle stampanti installate e consente all'utente di configurare o installare una nuova stampante di default. Per visualizzare la finestra di dialogo Stampa si deve utilizzare il metodo `ShowPrinter( )` e per usarla sarà necessario definire le appropriate impostazioni del controllo `CommonDialog` relative alla finestra di dialogo Stampa.

Ricordare. La finestra di dialogo Stampa è disponibile solo nella versione Professional di Visual Basic 6.

La tabella seguente riporta le proprietà più importanti della finestra di dialogo Stampa.

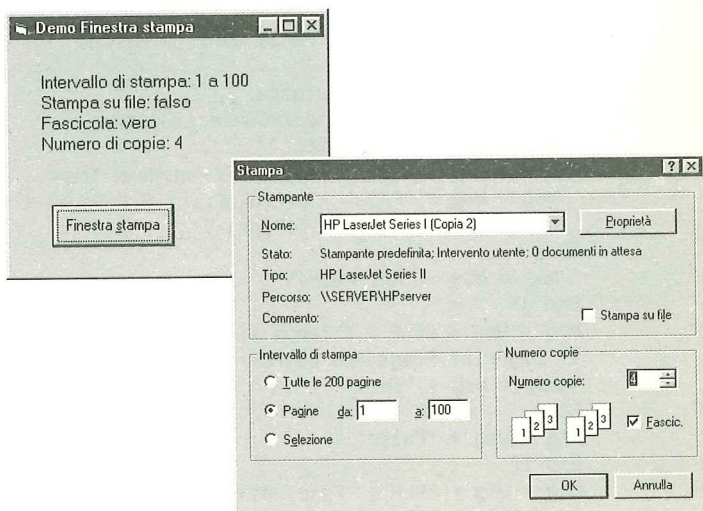
Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Copies: `oggetto.Copies [= numero]`	Restituisce o imposta il numero di copie da stampare.	`dlgPrint. Copies = 10`	Imposta il numero di copie da stampare a 10.
FromPage: `oggetto.FromPage [= numero]`	Restituisce o imposta il numero di pagina iniziale da stampare.	`dlgPrint. FromPage = 1`	Fa sì che la stampa inizi dalla prima pagina.
ToPage: `oggetto.ToPage [= numero]`	Restituisce o imposta il numero dell'ultima pagina da stampare.	`dlgPrint. ToPage = 10`	L'ultima pagina da stampare sarà la numero 10.
Min: `oggetto.Min [= numero]`	Restituisce o imposta l'intervallo minimo di pagine da stampare.	`dlgPrint. Min = 0`	Imposta a 0 l'intervallo minimo di pagine da stampare.
Max: `oggetto.Max [= numero]`	Restituisce o imposta l'intervallo massimo di pagine da stampare.	`dlgPrint. Max = 1000`	Imposta a 1000 l'intervallo massimo di pagine da stampare.
PrinterDefault: `oggetto.Printer Default [= booleano]`	Restituisce o imposta un'opzione che determina se le selezioni dell'utente della finestra di dialogo Stampa debbano essere applicate per modificare le impostazioni di sistema per la stampante predefinita.	`dlgPrint. Printer Default = True`	Applica le impostazioni di default alla finestra di dialogo Stampa.
Flags: `oggetto.Flags [= valore]`	Restituisce o imposta le opzioni per una finestra di dialogo Stampa (si veda la tabella seguente).	`dlgPrint. Flags = cdlAllPages`	Imposta la proprietà Flags a cdlAllPages.

La tabella seguente riporta i valori applicabili alla proprietà Flags.

Costante	*Valore*	*Descrizione*
cdlPDAllPages	**&H0**	Restituisce o imposta lo stato del pulsante di opzione per la stampa di tutte le pagine.
cdlPDCollate	**&H10**	Restituisce o imposta lo stato del pulsante di opzione per la fascicolazione delle pagine.
cdlPDDisable PrintToFile	**&H80000**	Disabilita la casella di controllo per la creazione di un file di stampa su disco.
cdlPDHelpButton	**&H800**	Inserisce nella finestra di dialogo il pulsante di comando per accedere alla guida.
cdlPDHideprint ToFile	**&H100000**	Nasconde la casella di controllo per la generazione di file di stampa su disco.
cdlPDNoPageNums	**&H8**	Disabilita il pulsante di opzione delle pagine da stampare e i corrispondenti controlli dipendenti.
cdlPDNoSelection	**&H4**	Disabilita il pulsante di opzione per la stampa di una selezione del documento.
cdlPDNoWarning	**&H80**	Evita la presentazione di messaggi quando non è stata impostata una stampante predefinita.
cdlPDPageNums	**&H2**	Restituisce o imposta lo stato del pulsante di opzione Pagine.
cdlPDPrintSetup	**&H40**	Consente al sistema di visualizzare la finestra di dialogo di impostazione della stampante al posto della finestra di dialogo Stampa.
cdlPDPrintToFile	**&H20**	Restituisce o imposta lo stato della casella di controllo per la generazione di stampa su disco.
cdlPDReturnDC	**&H100**	Restituisce un contesto di periferica relativo alla selezione di stampante effettuato all'interno della finestra di dialogo.
cdlPDReturn default	**&H400**	Restituisce il nome della stampante predefinita.
cdlPDReturnIC	**&H200**	Restituisce un'informazione contestuale relativa alla selezione della stampante fatta nella finestra di dialogo. Un'informazione contestuale è un sistema rapido per ottenere le informazioni su una periferica senza dovere creare un contesto di periferica.

continua

Costante	Valore	Descrizione
`cdlPDSelection`	&H1	Restituisce o imposta lo stato relativo al pulsante di opzione Selezione. Se non vengono specificati né `cdlPDPageNums` né `cdlPDSelection`, per default viene selezionato il pulsante di opzione che stampa tutte le pagine.
`cdlPDUseDevModeCopies`	&H40000	Se un driver di stampante non è in grado di gestire la stampa di copie multiple, l'attivazione di questa impostazione disabilita la casella per la definizione delle copie da stampare. In caso contrario, l'attivazione di questa impostazione indica che la finestra di dialogo memorizza il numero di copie richiesto all'interno della proprietà `Copies`.

La figura seguente è il risultato del programma di esempio che verrà ora presentato. Come si noterà, l'applicazione visualizza un modulo con un'etichetta e il pulsante di comando `Finestra stampa`. Quando l'utente farà clic su questo pulsante, il programma visualizza la finestra di dialogo Stampa. Quando l'utente chiude la finestra di dialogo, il programma visualizza, nel controllo Etichetta, l'intervallo di pagine da stampare, lo stato di stampa su disco e il numero di copie.

La tabella seguente riporta l'impostazione delle proprietà degli oggetti componenti la figura.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Demo Finestra stampa
Common Dialog	Name	dlgPrint
Pulsante comando	Name Caption	cmdPrintDialog Finestra &stampa
Etichetta	Name	lblPrintInfo

Ecco il codice sorgente.

```
Private Sub cmdprintDialog_Click( )
  Dim CRLF As String * 2
  Dim Msg As String

  CRLF = Chr$(13) + Chr$(10)
  ' svuota la modalità di default
  dlgPrint.PrinterDefault = False
  ' imposta la proprietà Flags
  dlgPrint.Flags = cdlPDAllPages Or cdlPDSelection
    Or cdlPDCollate Or cdlPDPageNums
  dlgPrint.Min = 1
  dlgPrint.Max = 200
  dlgPrint.FromPage = 1
  dlgPrint.ToPage = 100
  ' Visualizza la finestra di dialogo Stampa
  dlgPrint.ShowPrint
  Msg = "Intervallo di stampa: "
  If dlgPrint.Flags And cdlPDSelection Then
    Msg = Msg + Selezione"
  ElseIf dlgPrint.Flags And cdlPDPageNums Then
    Msg = Msg + "Page" + Str(dlgPrint.FromPage) +
    " to" + Str(dlgPrint.ToPage)
  Else
    Msg = Msg + "All Pages"
  End If
  Msg = Msg + CRLF + "Stampa su file: "
  If dlgPrint.Flags And cdlPDPrintToFile Then
    Msg = Msg + "Vero"
  Else
  Msg = Msg + "Falso"
  End If
  Msg = Msg + CRLF + "Fascicola: "
  If dlgPrint.Flags And cdlPDCollate Then
    Msg = Msg + "Vero"
  Else
```

```
    Msg = Msg + "Falso"
  End If
  Msg = Msg + CRLF + "Numero di copie:"
  + Str(dlgPrint.Copies)
  lblPrintInfo.Caption = Msg
End Sub

Private Sub Form_Load( )
  lblPrintInfo.Caption = ""
  lblPrintInfo.Font.Name = "Courier"
  lblPrintInfo.Font.Size = 12
End Sub
```

La procedura `Form_Load( )` inizializza il controllo Etichetta svuotandolo dal contenuto e impostando il tipo e la dimensione del carattere a Courier 12 punti.

La procedura `cmdPrintDialog_Click( )` risponde a un clic sul pulsante `Finestra stampa` eseguendo le operazioni seguenti.

- Imposta su `False` la proprietà `PrinterDefault`.
- Imposta la proprietà `Flags` della finestra di dialogo Stampa in modo da attivare i valori `cdlPDAllPages`, `cdlPDSelection`, `cdlPDCollate` e `cdlPDPageNums`. Questi valori consentono all'utente di selezionare la stampa di tutte le pagine, di stampare solo quelle selezionate o un loro intervallo e di decidere se attivare o meno la fascicolazione.
- Assegna 1 e 200 alle proprietà `Min` e `Max` della finestra di dialogo Stampa. Questa operazione definisce l'intervallo di pagine valido.
- Assegna 1 e 100 alle proprietà `FromPage` e `ToPage` della finestra di dialogo Stampa. Questa operazione definisce l'intervallo di pagine da stampare, così come riportato nella finestra di dialogo stessa.
- Visualizza la finestra di dialogo Stampa richiamata dal metodo `ShowPrint( )`.
- Determina il tipo di intervallo di pagine e ne memorizza il valore nella variabile stringa locale `Msg`. Questa operazione utilizza un enunciato `If-Else` che prende in esame la proprietà `Flags`. Se l'utente ha deciso di stampare un intervallo di pagine, l'operazione accede a tale intervallo tramite le proprietà `FromPage` e `ToPage`.
- Memorizza nella variabile `Msg` lo stato della casella di controllo per la creazione di una stampa su disco. Questa operazione si serve di un enunciato `If` che determina se l'utente ha attivato la casella di controllo per la stampa su disco.

- Memorizza nella variabile locale `Msg` lo stato della casella di controllo per la fascicolazione delle pagine servendosi di un enunciato `If` che determina se l'utente ha attivato tale casella di controllo.
- Memorizza nella variabile `Msg` il numero di copie da stampare. L'operazione accede alla proprietà `Copies` dalla quale estrae il numero di copie inserito dall'utente.
- Copia la stringa nella variabile `Msg` (che contiene varie righe di testo) e la visualizza all'interno del controllo "Etichetta" del form.

Controlli avanzati

In questa parte si prendono in esame i controlli avanzati di Visual Basic con i quali si può – quando combinati con i controlli basilari descritti nella *Parte II* – eseguire qualsiasi operazione. Molti dei controlli qui esaminati richiedono la versione Professional o Enterprise di Visual Basic 6 e, nella maggior parte dei casi, non sono disponibili nella versione Standard. I controlli qui descritti consentono di accedere a molte delle interfacce standard di Windows 9x tramite le quali si potranno aprire file, visualizzare immagini ed eseguire molte altre operazioni.

Argomenti trattati

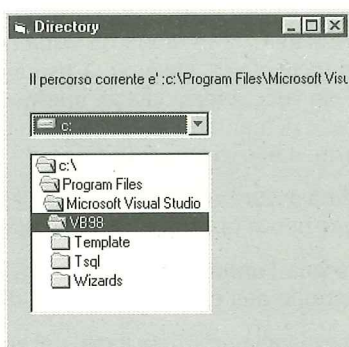
- ✔ **Uso dei controlli per visualizzare forme e immagini**
- ✔ **Uso dei controlli per consentire all'utente di accedere a file e cartelle**
- ✔ **Uso del temporizzatore per la gestione degli eventi temporizzati**

Controllo "DirListBox"

Visual Basic prevede la presenza del controllo Elenco cartelle (Directory List Box) con il quale, in fase di esecuzione dell'applicazione, si potrà visualizzare il contenuto e il percorso di file contenuti in cartelle (spesso chiamate directory) in base alla loro struttura gerarchica e di dipendenza.

La tabella seguente riporta le proprietà più importanti di questo controllo. Gli eventi più significativi sono, invece: `Click`, `Change`, Scroll, `GotFocus` e `LostFocus`.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Path: `oggetto.`**`Path`** `[= nome Percorso]`	Restituisce o imposta il percorso di memorizzazione attivo. Questa proprietà non è disponibile in fase di progettazione dell'applicazione.	`elencoDir. Path = "\Windows"`	Imposta il percorso di memorizzazione del controllo `elencoDir` in modo che venga visualizzato il contenuto della cartella Windows.
List: `oggetto.`**`List`** `(indice) [= stringa]`	Restituisce la voce della cartella selezionata all'interno del controllo.	`SelDir = elenco Dir.List(elenco Dir.ListIndex)`	Memorizza nella variabile SelDir la cartella selezionata.
ListCount: `oggetto.`**`ListCount`**	Restituisce il numero delle voci contenute nella cartella selezionata.	`If elencoDir. ListCount > 0 Then`	Determina quante voci sono contenute nel controllo `elencoDir`.
ListIndex: `oggetto.`**`ListIndex`** `[= indice]`	Restituisce o imposta l'indice della voce selezionata all'interno del controllo.	`miaSel = elencoDir.List (elencoDir. ListIndex)`	Memorizza nella variabile `mia Sel` la cartella selezionata

Si osservi ora la figura che è il risultato del programma di esempio che verrà ora presentato. Come si noterà, l'applicazione visualizza un form contenente un elenco di unità disco e di cartelle. Quando l'utente selezionerà una nuova unità disco o cartella, il programma rifletterà tale operazione segnalandola all'interno dell'etichetta del form stesso.

Nella tabella seguente si elencano le impostazioni delle proprietà degli oggetti componenti il Form.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Directory
Etichetta	Name	lblDrive
DirListBox	Name	drvDrive
DirListBox	Name	dirList

Ecco il codice sorgente.

```
Sub ShowPath( )
  lblDrive.Caption = "Il percorso corrente è " +
dirList.Path
End Sub
Private Sub DirList_Change( )
  ShowPath
End Sub
Private Sub drvDrive_Change( )
  drvDrive.Drive = drvDrive.List(drvDrive.ListIndex)
  dirList.Path = drvDrive.Drive
  ShowPath
End Sub
Private Sub Form_Load( )
  ShowPath
End Sub
```

Questo codice contiene la procedura `ShowPath( )` che visualizza, trasferendone il valore alla proprietà `Caption` del controllo Etichetta, il percorso attivo. La procedura `ShowPath()` accede a tale percorso servendosi della proprietà `Path` del controllo DirListBox.

La procedura `Form_Load( )` inizializza il form richiamando la procedura `ShowPath( )` in modo da visualizzare il percorso iniziale. La procedura `dirList_Change( )` risponde ai cambiamenti di selezione della voce selezionata all'interno della casella dell'elenco delle cartelle richiamando la procedura `ShowPath( )` in modo da visualizzarne il nuovo percorso.

La procedura `drvDrive_Change( )`, da parte sua, risponde alla selezione di una nuova unità disco eseguendo le seguenti operazioni.

- Assegna l'unità disco selezionata alla proprietà `Drive` del controllo dell'elenco delle unità disco.
- Assegna alla proprietà `Path` del controllo dell'elenco delle cartelle il valore della proprietà `Drive`.
- Visualizza il nuovo percorso richiamando la procedura `ShowPath( )`.

Controllo "DriveListBox"

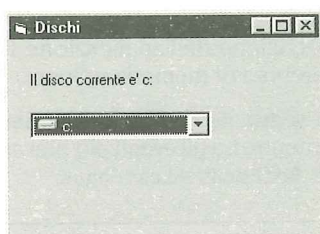
Il controllo Casella di riepilogo delle unità (Drive List Box) permette di selezionare, in fase di esecuzione dell'applicazione, un'unità disco. Questo controllo viene usato per visualizzare un elenco contenente tutte le unità disco installate nel sistema.

La tabella seguente riporta le più importanti proprietà applicabili a questo controllo. Gli eventi più significativi associati, sono, invece, `Click`, `Change`, `Scroll`, `GotFocus` e `LostFocus`.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Drive: `oggetto.`**`Drive`** `[= nomePercorso]`	Restituisce o imposta l'unità disco corrente. Questa proprietà non è disponibile in fase di progettazione dell'applicazione.	`drvDrive.Drive = "C:"`	Imposta l'unità disco del controllo `drvDrive` su C:
List: `oggetto.`**`List`** `(indice) [= stringa]`	Restituisce l'elenco delle voci del controllo DriveListBox.	`SelDrv = drvDrive.List (drvDrive.ListIndex)`	Memorizza nella variabile `SelDrv` l'unità disco selezionata.
ListCount: `oggetto.`**`ListCount`**	Restituisce il numero degli elementi contenuti nel controllo DriveListBox.	`If drvDrive.ListCount > 0 Then`	Determina la presenza di voci all'interno del controllo `drvDrive`

continua

Proprietà: sintassi	Scopo	Esempio	Commento
ListIndex: `oggetto.ListIndex [= indice]`	Restituisce o imposta l'indice della voce selezionata all'interno del controllo.	`miaSel = drvDrive.List (drvDrive. ListIndex)`	Memorizza nella variabile `miaSel` l'unità disco selezionata.

La figura seguente è il risultato del programma di esempio che verrà ora proposto. L'applicazione visualizza un form contenente un elenco di unità disco; quando l'utente selezionerà un'unità disco diversa, il programma rifletterà tale selezione all'interno del controllo Etichetta del modulo stesso.

La tabella seguente elenca l'impostazione delle proprietà dei vari oggetti del modulo.

Oggetto	Proprietà	Impostazione
Form	Name Caption	Form1 (default) Dischi
Etichetta	Name	lblDrive
DriveListBox	Name	drvDrive

Ecco il codice sorgente dell'applicazione.

```
Sub ShowDrive( )
  lblDrive.Caption = "Il disco corrente è " +
drvDrive.Drive
End Sub
Private Sub drvDrive_Change( )
  ShowDrive
End Sub
Private Sub Form_Load( )
  ShowDrive
End Sub
```

La procedura `ShowDrive( )` utilizza l'unità disco attiva assegnandone il valore corrispondente alla proprietà `Caption` del controllo "Etichetta". Questa routine accede all'unità disco tramite la proprietà `Drive` del controllo DriveListBox.

La procedura `Form_Load( )` inizializza il form richiamando la procedura `ShowPath( )` per visualizzare l'unità disco iniziale. La procedura `drvDrive_Change( )` risponde alla modifica dell'unità disco all'interno del controllo DriveListBox richiamando la procedura `ShowPath( )` visualizzando, così, il nuovo identificativo di unità disco.

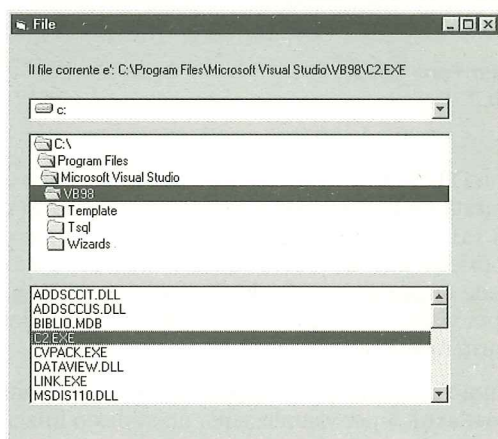
Controllo FileListBox

Il controllo FileListBox di Visual Basic consente di visualizzare, in fase di esecuzione dell'applicazione, l'elenco dei file di un determinato percorso di memorizzazione.

La tabella seguente riporta le proprietà più importanti di questo controllo, mentre tra gli eventi più rilevanti, si ricordano `Click`, `Change`, `Scroll`, `GotFocus` e `LostFocus`.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Path: `oggetto.Path [= nomepercorso]`	Restituisce o imposta il percorso attivo. Questa proprietà non è disponibile in fase di progettazione dell'applicazione.	`fileList.Path = "C:"`	Imposta il percorso del controllo `fileList` alla cartella Windows dell'unità disco C:
List: `oggetto.List (indice) [= stringa]`	Restituisce l'elenco di voci contenute nel controllo `File List Box`.	`SelFile = fileList.List(fileList.ListIndex)`	Memorizza nella variabile `SelFile` il file selezionato.
ListCount: `oggetto.ListCount`	Restituisce il numero di elementi contenuti nel controllo `File List Box`.	`If fileList.ListCount > 0 Then`	Determina se vi sono elementi nel controllo `fileList`.
ListIndex: `oggetto.ListIndex [= indice]`	Restituisce o imposta l'indice della voce selezionata all'interno del controllo.	`miaSel = fileList.List(fileList.ListIndex)`	Memorizza il file selezionato all'interno della variabile `miaSel`.
Filename: `oggetto.Filename`	Restituisce il nome del file selezionato.	`SelFile = filList.Filename`	Memorizza il nome del file selezionato all'interno della variabile `SelFile`.

Per maggiori informazioni sul controllo File List Box, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

La figura seguente è il risultato del programma di esempio che verrà ora presentato. Come si noterà, la figura è composta da un form che contiene le caselle per la visualizzazione dell'unità disco, della cartella e dell'elenco dei file. Quando l'utente selezionerà una nuova unità disco, cartella o file, il programma rifletterà tale scelta all'interno del testo dell'etichetta contenuta nel modulo stesso.

La tabella seguente riporta l'elenco delle impostazioni delle proprietà degli oggetti del form.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) File
Etichetta	Name	lblDrive
DriveListBox	Name	drvDrive
DirListBox	Name	dirList
FileListBox	Name	filList

Ecco il codice sorgente dell'applicazione.

```
Sub ShowPath(BackSlash As String)
  lblDrive.Caption = "Il file corrente è " +
dirList.List(dirList.ListIndex) + BackSlash +
filList.Filename
End Sub
Private Sub dirList_Change( )
  filList.Filename = dirList.Path
```

```
  ShowPath " "
End Sub
Private Sub drvDrive_Change( )
  drvDrive.Drive = drvDrive.List(drvDrive.ListIndex)
  dirList.Path = drvDrive.Drive
  ShowPath " "
End Sub
Private Sub filList_Click( )
  ShowPath "\"
End Sub
Private Sub Form_Load( )
  ShowPath " "
End Sub
```

La procedura `ShowPath( )` visualizza il percorso attivo e il file selezionato. Questa routine accede al percorso servendosi dell'espressione `dirList.List(dirList.ListIndex)`e al nome del file tramite la proprietà `Filename` del controllo `File List Box`. La procedura prevede un argomento `BackSlash` di tipo `String` che viene utilizzato per visualizzare, quando necessario, prima del nome del file, una barra rovesciata ("\").

La procedura `Form_Load( )` inizializza il form richiamando la procedura `ShowPath( )` per visualizzare l'unità disco iniziale. Il valore dell'argomento di questa chiamata è pari a una stringa contenente uno spazio (`" "`). La procedura `filList_Click( )` risponde a un clic all'interno del controllo `File List Box` richiamando la procedura `ShowPath( )` per visualizzare il percorso di memorizzazione corrente e il nuovo file selezionato. Il valore dell'argomento di questa chiamata corrisponde alla stringa `"\"`.

La procedura `dirList_Change( )` risponde alla selezione di una nuova cartella assegnando alla proprietà `Filename` del controllo File List Box il valore della proprietà `Path` del controllo DirListBox e richiamando, successivamente, la procedura `ShowPath( )`. Il valore dell'argomento per la chiamata della procedura `ShowPath( )` corrisponde a una stringa contenente uno spazio (`" "`).

La procedura `drvDrive_Change( )` risponde alla selezione di una nuova unità disco eseguendo le operazioni seguenti.

- Assegna l'unità disco selezionata alla proprietà `Drive` del controllo DriveListBox.
- Assegna la proprietà `Drive` del controllo DriveListBox alla proprietà `Path` del controllo DirListBox.
- Richiama la procedura `ShowPath( )` utilizzando una stringa contenente uno spazio (`" "`) come valore di argomento della chiamata stessa.

Controllo "Immagine"

Il controllo Immagine (Image) consente di visualizzare della grafica o un'icona all'interno di un form. L'immagine sorgente deve essere di tipo bitmap, un'icona, un'immagine in formato metafile o enhanced metafile, JPEG o GIF. Sebbene si abbia la possibilità di inserire un controllo Immagine in una struttura di contenitore (tipo di controllo particolare di Visual Basic che permette di racchiudere e raggruppare altri controlli tra loro correlati), un controllo Immagine non può funzionare come contenitore.

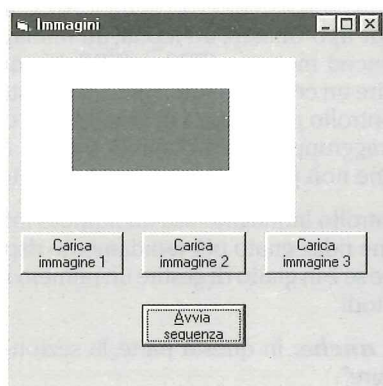
Il controllo Immagine usa un numero inferiore di risorse di sistema e viene ridisegnato più rapidamente rispetto a un controllo Picture anche se è in grado di gestire un numero inferiore di proprietà, eventi e metodi.

Vedi anche: in questa parte, la sezione che descrive il controllo "Picture".

La tabella seguente riporta le proprietà più importanti di un controllo Immagine mentre gli eventi più rilevanti sono `Click`, `DblClick` e quelli correlati alle operazioni eseguite con il mouse.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Picture: `oggetto.`**`Picture`** `[= immagine]`	Restituisce o imposta il contenuto grafico del controllo `Immagine`.	`imgPic.Picture = LoadPicture ("io.bmp")`	Carica il contenuto grafico del file `IO.BMP` e lo inserisce all'interno del controllo `imgPic`.
Stretch: `oggetto.`**`Stretch`** `[= booleano]`	Restituisce o imposta un valore che indica se l'immagine grafica deve essere ridimensionata per adattarsi alle dimensioni del controllo `Immagine`.	`imgPic.Stretch = True`	Imposta la proprietà `Stretch` del controllo immagine `imgPic`.

Viene ora proposto un esempio di applicazione che utilizza un controllo Immagine e il cui risultato viene visualizzato nella figura seguente. L'esempio genera un form contenente un controllo Immagine e quattro pulsanti. I primi tre richiamano l'immagine grafica memorizzata nei file PIC1.BMP, PIC2.BMP e PIC3.BMP mentre il quarto fa sì che le tre immagini grafiche contenute in questi file vengano visualizzate sequenzialmente come se si trattasse di una presentazione a video. Il programma fa uso di un controllo di temporizzazione per visualizzare ogni immagine grafica per un secon-

do e quindi richiamare l'immagine successiva, visualizzandola per un altro secondo e così via.

La tabella seguente elenca le impostazioni delle proprietà degli oggetti del modulo.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Immagini
Controllo Immagine	Name	imgBMP
Pulsante di comando	Name Caption	cmdBMP1 Carica immagine 1
Pulsante di comando	Name Caption	cmdBMP2 Carica immagine 2
Pulsante di comando	Name Caption	cmdBMP3 Carica immagine 3
Pulsante di comando	Name Caption	cmdSlide &Avvia sequenza
Timer	Name Interval	tmrTimer 1000

Ecco il codice sorgente dell'applicazione.

```
Dim ImageIndex As Integer
Private Sub cmdBmp1_Click( )
  Set imgBmp.Picture = LoadPicture("\pic1.bmp")
End Sub
Private Sub cmdBmp2_Click( )
  Set imgBmp.Picture = LoadPicture("\pic2.bmp")
```

```
End Sub
Private Sub cmdBmp3_Click( )
  Set imgBmp.Picture = LoadPicture("\pic3.bmp")
End Sub
Private Sub cmdSlide_Click( )
  If cmdSlide.Caption = "&Avvia sequenza" Then
    cmdSlide.Caption = "&Ferma sequenza"
    ImageIndex = 1
    tmrTimer.Enabled = True
  Else
    cmdSlide.Caption = "&Avvia sequenza"
    tmrTimer.Enabled = False
  End If
End Sub
Private Sub Form_Load( )
  imgBmp.Stretch = True
  tmrTimer.Enabled = False
End Sub
Private Sub tmrTimer_Timer( )
  Set imgBmp.Picture = LoadPicture("\pic" +
Mid(Str(ImageIndex), 2) + ".bmp")
  ImageIndex = ImageIndex Mod 3 + 1
End Sub
```

Questo codice dichiara la variabile `ImageIndex` di tipo intero e a livello di form in modo da conservare una traccia della sequenza delle immagini visualizzate quando si richiama la procedura per la presentazione sequenziale delle immagini stesse.

La procedura `Form_Load( )` inizializza il form impostando la proprietà `Stretch` del controllo Immagine su `True` e disabilitando, nel contempo il controllo del temporizzatore.

La procedura `cmdBmp1_Click( )` risponde a un clic sul pulsante di comando `Carica immagine 1` caricando l'immagine grafica del file PIC1.BMP (memorizzata nella cartella radice). Le procedura `cmdBmp2_Click( )` e `cmdBmp3_Click()` si comportano nello stesso modo caricando, rispettivamente, le immagini grafiche dei file PIC2.BMP e PIC3.BMP ogni volta che si fa clic sui pulsanti `Carica immagine 2` e `Carica immagine 3`.

La procedura `cmdSlide_Click( )` risponde al clic sul pulsante `Avvia sequenza` esaminando il contenuto della proprietà `Caption` del pulsante di comando per determinare lo stato della funzione di visualizzazione sequenziale temporizzata delle immagini. Quando la proprietà `Caption` del pulsante di comando è impostata su `Avvia sequenza` il gestore di evento esegue le operazioni seguenti.

- Imposta la proprietà `Caption` del pulsante di comando su `Ferma sequenza`.

- Assegna il valore 1 alla variabile `ImageIndex`.
- Abilita il controllo del temporizzatore.

Quando, invece, la proprietà `Caption` del pulsante di comando è impostata su `Ferma sequenza` il gestore di evento esegue le seguenti operazioni.

- Imposta la proprietà `Caption` del pulsante di comando su `Avvia sequenza`.
- Disabilita il controllo del temporizzatore.

La procedura `tmrTimer_Timer( )` risponde all'evento di temporizzazione eseguendo le seguenti operazioni.

- Imposta la proprietà `Picture` del controllo Immagine inserendovi una nuova immagine grafica. Questa operazione si serve della procedura `LoadPicture` che carica il file .BMP corrispondente. Il nome del file .BMP utilizza la stringa del valore della variabile `ImageIndex`.
- Aggiorna il valore della variabile `ImageIndex` assegnandogli il valore successivo (compreso nell'intervallo tra 1e 3).

Controllo "PictureBox"

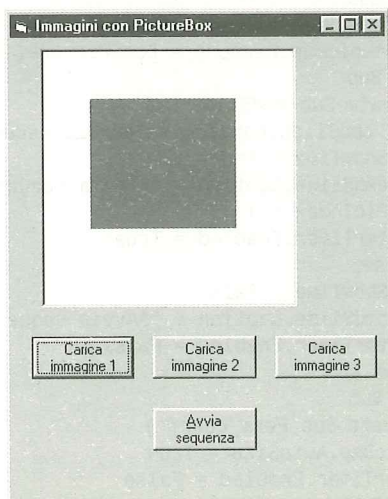
Un controllo Picture Box è in grado di visualizzare un'immagine grafica di tipo bitmap, un'icona, un metafile, un metafile enhanced o file di tipo JPEG o GIF. Il controllo ritaglia l'immagine all'interno del controllo nel caso in cui l'immagine stessa sia di dimensioni superiori a quelle definite dalla struttura di contenimento. Si ha anche la possibilità di utilizzare il controllo in questione per controllare un gruppo di pulsanti di opzione e per visualizzare il risultato dei metodi per la visualizzazione delle immagini grafiche e del testo, gestiti dal metodo `Print( )`.

Il controllo PictureBox utilizza una quantità superiore di risorse di sistema e ridisegna le immagini più lentamente di quanto accade con l'uso del controllo Immagine ma, per contro, è in grado di accedere a un numero superiore di proprietà, eventi e metodi.

Vedi anche: in questa parte, la sezione che descrive il controllo "Immagine".

Nella tabella seguente vengono elencate le proprietà più importanti del controllo PictureBox mentre gli eventi più rilevanti per lo stesso controllo sono `Click`, `DblClick` e tutti gli eventi correlati all'uso del mouse.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Picture: `oggetto.Picture [= immagine]`	Restituisce o imposta l'immagine da inserire in un controllo PictureBox.	`picImage.Picture = LoadPicture ("io.bmp")`	Carica un'immagine grafica prelevandola dal file `io.bmp` e la inserisce nel controllo picImage.
AutoSize: `oggetto.AutoSize [= booleano]`	Restituisce o imposta il valore che determina se il controllo deve automaticamente ridimensionare l'immagine per visualizzarla completamente all'interno del controllo stesso.	`picImage.AutoSize = True`	Imposta la proprietà `AutoSize` perché ridimensioni l'immagine inserita nel controllo picImage.

Il programma che viene ora proposto, il cui risultato finale è riportato nella figura, visualizza un form contenente un controllo PictureBox e quattro pulsanti. I primi tre visualizzano, rispettivamente, le immagini contenute nei file PIC1.BMP, PIC2.BMO e PIC3.BMP, mentre il quarto avvia una presentazione sequenziale delle tre immagine con un intervallo di visualizzazione temporizzata di 1 secondo ciascuna. Lo stesso pulsante viene utilizzato anche per interrompere la presentazione sequenziale delle tre immagini. Il programma utilizza, in quest'ultimo caso, un temporizzatore che memorizza e visualizza l'immagine per un secondo prima di visualizzare la successiva.

La tabella seguente riporta le impostazioni delle proprietà degli oggetti del modulo.

Oggetto	*Proprietà*	*Impostazione*
Form	Name Caption	Form1 (default) Immagini con PictureBox
PictureBox	Name	picBmp
Pulsante di comando	Name Caption	cmdPic1 Carica immagine 1
Pulsante di comando	Name Caption	cmdPic2 Carica immagine 2
Pulsante di comando	Name Caption	cmdPic3 Carica immagine 3
Pulsante di comando	Name Caption	cmdSlide &Avvia sequenza
Timer	Name Interval	tmrTimer 1000

Ecco il codice sorgente dell'applicazione.

```
Dim PicIndex As Integer
Private Sub cmdPic1_Click( )
  Set picBmp.Picture = LoadPicture("\pic1.bmp")
End Sub
Private Sub cmdPic2_Click( )
  Set picBmp.Picture = LoadPicture("\pic2.bmp")
End Sub
Private Sub cmdPic3_Click( )
  Set picBmp.Picture = LoadPicture("\pic3.bmp")
End Sub
Private Sub cmdSlide_Click( )
  If cmdSlide.Caption = "&Avvia sequenza" Then
    ShowTime = True
    cmdSlide.Caption = "&Ferma sequenza"
    PicIndex = 1
    tmrTimer.Enabled = True
  Else
    ShowTime = False
    cmdSlide.Caption = "&Avvia sequenza"
    tmrTimer.Enabled = False
  End If
End Sub
Private Sub Form_Load( )
  picBmp.AutoSize = True
  tmrTimer.Enabled = False
End Sub
```

```
Private Sub tmrTimer_Timer( )
  Set picBmp.Picture = LoadPicture("\pic" +
Mid(Str(PicIndex), 2) + ".bmp")
  PicIndex = PicIndex Mod 3 + 1
End Sub
```

Questo codice dichiara la variabile `PicIndex` di tipo intero a livello di form in modo da conservare una traccia della sequenza delle immagini visualizzate quando si richiama la procedura per la presentazione sequenziale delle immagini stesse.

La procedura `Form_Load( )` inizializza il form impostando la proprietà AutoSize del controllo PictureBox su `True` e disabilitando, nel contempo il controllo del temporizzatore.

La procedura `cmdPic1_Click( )` risponde a un clic sul pulsante di comando `Carica immagine 1` caricando l'immagine grafica del file PIC1.BMP (memorizzata nella cartella radice).

Le procedura `cmdPic2_Click( )` e `cmdPic3_Click( )` si comportano nello stesso modo caricando, rispettivamente, le immagini grafiche dei file PIC2.BMP e PIC3.BMP ogni volta che si fa clic sui pulsanti `Carica immagine 2` e `Carica immagine 3`.

La procedura `cmdSlide_Click( )` risponde al clic sul pulsante `Avvia sequenza` esaminando il contenuto della proprietà Caption del pulsante di comando per determinare lo stato della funzione di visualizzazione sequenziale temporizzata delle immagini. Quando la proprietà Caption del pulsante di comando è impostata su `&Avvia sequenza` il gestore di evento esegue le operazioni seguenti.

- Imposta la proprietà `Caption` del pulsante di comando su `Ferma sequenza`.
- Assegna il valore 1 alla variabile `PicIndex`.
- Abilita il controllo del temporizzatore.

Quando, invece, la proprietà `Caption` del pulsante di comando è impostata su `&Ferma sequenza` il gestore di evento esegue le seguenti operazioni.

- Imposta la proprietà `Caption` del pulsante di comando su `Avvia sequenza`.
- Disabilita il controllo del temporizzatore.

La procedura `tmrTimer_Timer( )` risponde all'evento di temporizzazione eseguendo le seguenti operazioni.

- Imposta la proprietà `Picture` del controllo PictureBox inserendovi una nuova immagine grafica. Questa operazione si

serve della procedura `LoadPicture( )` che carica il file .BMP corrispondente. Il nome del file .BMP utilizza la stringa del valore della variabile `ImageIndex`.

- Aggiorna il valore della variabile `PicIndex` assegnandogli il valore successivo (compreso nell'intervallo tra 1e 3).

Controllo "Forma"

Il controllo Forma (Shape) è un controllo di tipo grafico visualizzato sotto forma di rettangolo, quadrato, ellisse, cerchio, rettangolo con angoli arrotondati o quadrato con angoli arrotondati. L'uso di questo controllo consente, quindi, di inserire rapidamente in un form uno di questi elementi.

La tabella seguente riporta le proprietà più importanti del controllo Forma mentre gli eventi più rilevanti sono `Click`, `DblClick` e tutti gli eventi collegati all'uso del mouse.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Shape: `oggetto.`**`Shape`** `[= numero]`	Restituisce o imposta la forma da utilizzare con il controllo.	`tipoForma. Shape = 0`	Assegna al controllo il valore `0` (corrispondente a un rettangolo).
FillColor: `oggetto.`**`FillColor`** `[= numero colore]`	Restituisce o imposta il colore di riempimento della forma.	`tipoForma. FillColor = vbRed`	Imposta la proprietà `FillColor` dell'oggetto disegnato sul rosso.
FillStyle: `oggetto.`**`FillStyle`** `[= numero]`	Restituisce o imposta il motivo di riempimento da applicare all'oggetto realizzato con il controllo Forma (vedi la tabella successiva).	`tipoForma. FillStyle = vbFSSolid`	Imposta la proprietà `FillStyle` dell'oggetto disegnato su un riempimento pieno.

La tabella seguente riporta i valori ammissibili per la proprietà `Shape` di un oggetto realizzato con il controllo Forma.

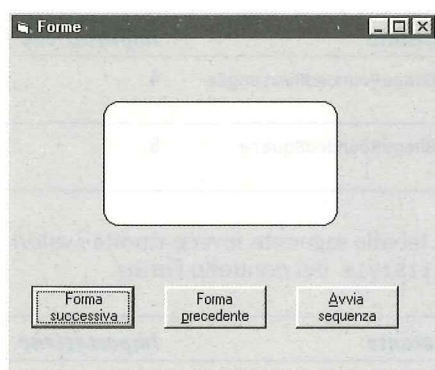
Costante	*Impostazione*	*Descrizione*
`vbShapeRectangle`	0	Crea un rettangolo (default)
`vbShapeSquare`	1	Crea un quadrato
`vbShapeOval`	2	Crea un'ellisse
`vbShapeCircle`	3	Crea un cerchio

continua

Costante	*Impostazione*	*Descrizione*
`vbShapeRoundedRectangle`	4	Crea un rettangolo con angoli arrotondati
`vbShapeRoundedSquare`	5	Crea un quadrato con angoli arrotondati

La tabella seguente, invece, riporta i valori ammessi dalla proprietà `FillStyle` del controllo Forma.

Costante	*Impostazione*	*Descrizione*
`vbFSSolid`	0	Applica un riempimento pieno
`vbFSTransparent`	1	Applica un riempimento trasparente (default)
`vbHorizontalLine`	2	Applica un riempimento composto da linee orizzontali
`vbVerticalLine`	3	Applica un riempimento composto da linee verticali
`vbUpwardDiagonal`	4	Applica un riempimento con linee diagonali verso l'alto
`vbDownwardDiagonal`	5	Applica un riempimento con linee diagonali verso il basso
`vbCross`	6	Applica un riempimento a croci
`vbDiagonalCross`	7	Applica un riempimento a croci diagonali

L'esempio di programma che si propone utilizza il controllo Forma per generare un form simile a quello della figura seguente. Il form generato contiene un controllo Forma e tre pulsanti. I primi due visualizzano la forma successiva e precedente (le forme sono un rettangolo, un quadrato, un'ellisse, un cerchio, un rettangolo con angoli arrotondati e un quadrato con angoli arrotondati). Il terzo pulsante, invece, avvia una procedura di visualizzazione sequenziale temporizzata delle forme create.

L'applicazione si serve di un controllo temporizzatore per visualizzare una forma per un secondo, quella successiva per un altro secondo, e così via.

La tabella seguente riporta le impostazioni delle proprietà degli oggetti del modulo.

Oggetto	Proprietà	Impostazione
Form	Name Caption	Form1 (default) Forme
Controllo "Forma"	Name	shpBody
Pulsante comando	Name Caption	cmdNext Forma &successiva
Pulsante comando	Name Caption	cmdPrev Forma &precedente
Pulsante comando	Name Caption	cmdSlide &Avvia sequenza
Controllo temporizzatore	Name Interval	tmrTimer 1000

Il codice sorgente dell'applicazione è il seguente.

```
DimFillColorArr(0 To 5) As Long
Private Sub cmdNext_Click( )
  shpBody.Shape = (shpBody.Shape + 1) Mod 6
  shpBody.FillColor = FillColorArr(shpBody.Shape)
End Sub
Private Sub cmdPrev_Click( )
  If shpBody.Shape > 0 Then
    shpBody.Shape = shpBody - 1
  Else
    shpBody.Shape = 5
```

```
  End If
  shpBody.FillColor = FillColorArr(shpBody.Shape)
End Sub
Private Sub cmdSlide_Click( )
  If cmdSlide.Caption = "&Avvia sequenza" Then
    cmdSlide.Caption = "&Ferma sequenza"
    shpBody.Shape = 0
    tmrTimer.Enabled = True
  Else
    cmdSlice.Caption = "&Avvia sequenza"
    tmrTimer.Enabled = False
  End If
End Sub
Private Sub Form_Load( )
  tmrTimer.Enabled = False
  shpBody.SillStyle = vbFSTransparent
  shpBody.Shape = 0
  FillColorArr(0) = vbCyan
  FillColorArr(1) = vbMagenta
  FillColorArr(2) = vbRed
  FillColorArr(3) = vbBlue
  FillColorArr(4) = vbYellow
  FillColorArr(5) = vbGreen
  shpBody.FillColor = FillColorArr(shpBody.Shape)
End Sub
Private Sub tmrTimer_Timer( )
  shpBody.Shape = (shpBody.Shape + 1) Mod 6
  shpBody.FillColor = FillColorArr(shpBody.Shape)
End Sub
```

Questo codice dichiara la matrice `FillColorArr( )` di tipo long a livello di form, per memorizzarvi i colori di riempimento delle varie forme che verranno utilizzate dal programma.

La procedura `Form_Load( )` inizializza il modulo eseguendo le operazioni seguenti.

- Disabilita il controllo di temporizzazione.
- Imposta la proprietà `FillStyle` del controllo Forma assegnandogli il valore `vbFSSolid` per generare un riempimento pieno.
- Imposta la proprietà `Shape` del controllo Forma impostandola a 0. Questo valore genera un rettangolo.
- Assegna agli elementi della matrice `FillColorArr( )` le costanti dei colori Visual Basic.
- Imposta la proprietà `FillColor` del controllo Formaal colore specificato dall'elemento 0 della matrice `FillColorArr( )`.

La procedura `cmdNext_Click( )` risponde al clic sul pulsante `Forma successiva` selezionando la forma successiva incrementando il valore della proprietà `Shape` del controllo Forma. Il gestore di evento si serve dell'operatore `Mod` per assicurare che la proprietà `Shape` sia compresa nell'intervallo che va da 0 a 5. Viene inoltre assegnato un nuovo valore alla proprietà `FillColor` del controllo Forma (questo nuovo valore utilizza la matrice `FillColorArr( )` e, per definire l'indice della matrice, il valore della proprietà `Shape`).

La procedura `cmdPrev_Click( )` risponde al clic sul pulsante `Forma precedente` selezionando la forma precedente, diminuendo il valore della proprietà `Shape` del controllo Forma. L'evento utilizza un enunciato `If` per assicurare che la proprietà `Shape` venga contenuta nell'intervallo di valori compreso tra 0 e 5. Il gestore di evento assegna anche un valore alla proprietà `FillColor` del controllo Forma (questo nuovo valore utilizza la matrice `FillColorArr( )` e, come indice per la matrice, il valore della proprietà `Shape`).

La procedura `cmdSlide_Click( )` risponde al clic sul pulsante `Avvia sequenza` esaminando il valore della proprietà `Caption` del pulsante di comando per determinare lo stato della funzione di visualizzazione sequenziale temporizzata. Quando la proprietà `Caption` del pulsante di comando contiene il valore `&Avvia sequenza` il gestore di evento esegue le operazioni seguenti.

- Imposta la proprietà `Caption` del pulsante di comando su `&Ferma sequenza`.
- Assegna il valore 0 alla proprietà `Shape` del controllo Forma.
- Abilita il controllo di temporizzazione.

Quando la proprietà `Caption` contiene il valore `&Ferma sequenza`, il gestore di evento esegue le seguenti operazioni.

- Imposta la proprietà `Caption` su `&Avvia sequenza`.
- Disabilita il controllo del temporizzatore.

La procedura `tmrTimer_Timer( )` risponde all'evento di temporizzazione eseguendo le operazioni seguenti.

- Seleziona la forma successiva incrementando il valore della proprietà `Shape` del controllo Forma. Il gestore di evento si serve, per questa operazione, dell'operatore `Mod` per garantire che il valore della proprietà `Shape` sia compreso nell'intervallo tra 0 e 5.
- Assegna un nuovo valore alla proprietà `FillColor` del controllo Forma e tale nuovo valore utilizza la matrice `FillCo-`

lorArr() e, per definire l'indice della matrice stessa, il valore della proprietà Shape.

Controllo "Temporizzatore"

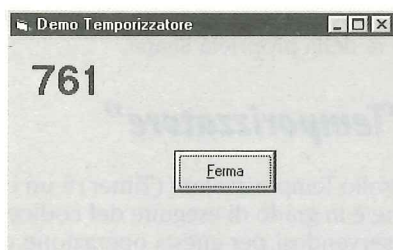
Il controllo Temporizzatore (Timer) è un controllo di tipo non visibile che è in grado di eseguire del codice a intervalli di tempo regolari servendosi, per questa operazione, dell'evento Timer. Il controllo di temporizzazione si rivela particolarmente utile per l'esecuzione di operazioni in background. La proprietà Enabled di questo controllo permette di attivarne o disattivarne l'operatività.

Quando la proprietà Enabled è impostata su True, la proprietà Interval specifica l'intervallo temporale nel quale il controllo di temporizzazione deve generare un successivo evento Timer.

Nota. Visual Basic 6 non presenta, virtualmente, alcuna limitazione per quanto riguarda il numero di controlli di temporizzazione che si possono inserire in un programma.

La tabella seguente riporta le proprietà più importanti che si possono associare a un controllo Temporizzatore; l'evento più rilevante è, invece, Timer.

Proprietà: sintassi	*Scopo*	*Esempio*	*Commento*
Enabled: oggetto.**Enabled** [= booleano]	Restituisce o imposta lo stato di abilitazione o disabilitazione del controllo Temporizzatore.	tmrTick. Enabled = False	Disabilita il controllo di temporizzazione tmrTick.
Interval: oggetto.**Interval** [= millisecondi]	Restituisce o imposta l'intervallo di tempo (espresso in millisecondi) entro il quale deve essere avviato un successivo evento Timer.	tmrTick. Interval = 2000	Imposta l'intervallo di temporizzazione del controllo tmrTick a 2 secondi.

L'esempio che viene ora proposto si serve di un controllo di temporizzazione che genera un risultato simile a quello di figura seguente. In questo esempio viene visualizzato un form contenente un'etichetta e il pulsante di comando Avvia. Quando l'utente fa clic sul pulsante di comando, il programma visualizza, ogni due secondi, una serie di numeri casuali. Quando la temporizzazione è attiva, il pulsante di comando Avvia viene convertito in Ferma. Se si fa clic su questa versione del pulsante di comando, si interromperà la generazione di numeri casuali.

La tabella seguente riporta le impostazioni delle proprietà degli oggetti che fanno parte del form.

Oggetto	Proprietà	Impostazione
Form	Name Caption	Form1 (default) Demo Temporizzatore
Controllo temporizzatore	Name Interval	tmrTimer 2000
Pulsante comando	Name Caption	cmdStart &Avvia
Etichetta	Name	lblNumber

Ecco il codice sorgente che genera il risultato finale desiderato.

```
Option Explicit
Private Sub cmdStart_Click( )
  If cmdStart.Caption = "&Avvia" Then
    cmdStart.Caption = "&Ferma"
    tmrTimer.Enabled = True
  Else
    cmdStart.Caption = "&Avvia"
    tmrTimer.Enabled = False
  End If
End SubPrivate Sub Form_Load( )
  tmrTimer.Enabled = False
  lblNumber.Caption = "0"
  òlblNumber.FontSize = 24
End Sub
Private Sub tmrTimer_Timer( )
  lblNumber.Caption = Format(1000 * Rnd(1), "###"
End Sub
```

La procedura `Form_Load( )` inizializza il form disabilitando il controllo di temporizzazione, svuotando il contenuto dell'etichetta e impostando il tipo di carattere della stessa a 24 punti.

La procedura cmdStart_Click( ) risponde al clic sul pulsante di comando Avvia esaminando il valore della proprietà Caption del pulsante di comando. Quando questa proprietà è impostata su &Avvia, il gestore di evento esegue le seguenti operazioni.

- Imposta la proprietà Caption del pulsante di comando su &Ferma.
- Abilita il controllo del temporizzatore.

Quando la proprietà Caption del pulsante di comando è impostata su &Ferma, il gestore di evento esegue le operazioni seguenti.

- Imposta la proprietà Caption del pulsante di comando su &Avvia.
- Disabilita il controllo di temporizzazione.

La procedura tmrTimer_Timer( ) risponde all'evento Timer visualizzando un numero casuale all'interno dell'etichetta del form.

Parte VI

Gestione degli errori

La gestione degli errori e la presa di decisioni sono due lati della programmazione che rendono piacevole la progettazione delle applicazioni e che consentono di dare loro una certa personalità. A conferma di ciò si pensi che ogni volta che si scrive un programma, *se* (`If`) questo non ne vuole sapere di funzionare, *allora* (`Then`) è necessario andare alla ricerca di possibili errori nel suo codice sorgente.

Argomenti trattati

- ✔ **Gestione degli errori che possono verificarsi quando si esegue il programma**
- ✔ **Come sopravvivere quando si presenta un errore**
- ✔ **Far prendere al programma le decisioni in modo che funzioni correttamente**

Oggetto Err

L'oggetto `Err` memorizza tutte le informazioni relative a errori che dovessero verificarsi nel corso dell'esecuzione di un'applicazione. Le proprietà dell'oggetto `Err` vengono impostate da un *generatore di errori* che potrebbe corrispondere a Visual Basic stesso, a un oggetto oppure a un enunciato di programma. Quando si verifica un errore di esecuzione, le proprietà dell'oggetto `Err` memorizzeranno le informazioni che consentiranno al programmatore di gestire l'errore che si è verificato.

Vedi anche: in questa parte, la sezione che descrive l'enunciato "Resume".

Le proprietà di un oggetto `Err` vengono reimpostate automaticamente a zero o a una stringa vuota (`""`) dopo ogni esecuzione di un enunciato `Resume` o `Error` e dopo ogni enunciato `Exit Sub`, `Exit Function` o `Exit Property` contenenti una routine per la gestione degli errori. Usare il metodo `Raise( )` per generare errori di esecuzione in un modulo di classe. L'uso e la sintassi del metodo `Raise( )` all'interno del codice dipendono dal tipo e quantità di informazioni che si desiderano ottenere su un errore.

La sintassi generale del metodo `Raise( )` è la seguente.

```
Err.Raise numero, origine, descrizione, fileGuida,
contestoGuida
```

Gli argomenti del metodo `Raise( )` corrispondono alle proprietà dell'oggetto Err, schematizzate nella tabella seguente.

Proprietà	Valore
`Number`	Valore che identifica il tipo di errore. Può corrispondere a un numero di errore (compreso tra 0 e 65535, come descritto nella tabella seguente).
`Source`	Nome del progetto Visual Basic nel quale si è verificato l'errore.
`Description`	Stringa corrispondente all'errore con la specifica del valore della proprietà `Number` (sempre che questo sia disponibile). In caso contrario la proprietà `Description` contiene la stringa `"Errore definito dall'applicazione o dall'oggetto"`.
`HelpFile`	Il percorso completo e il nome del file di guida di Visual Basic.
`HelpContext`	L'identificativo del file di guida contestuale di Visual Basic corrispondente al valore della proprietà `Number`.

L'oggetto `Err` sostituisce l'enunciato `Error` delle versioni precedenti di Visual Basic. Sebbene questo enunciato sia ancora disponibile anche nella versione 6, si consiglia di non utilizzarlo e di preferire l'oggetto `Err`.

La tabella seguente riporta l'elenco dei più comuni errori che si verificano nel corso dell'esecuzione di un'applicazione.

Codice	Messaggio emesso
5	Chiamata di procedura non valida o argomenti non validi.
6	Errore di `overflow`.
7	Memoria esaurita.
9	Indice non compreso nell'intervallo.
10	Questa matrice è di dimensione fissa o temporaneamente bloccata.
11	Divisione per zero.
13	Tipo non concordante.
14	Spazio stringa esaurito.
16	Espressione troppo complessa.
20	Resume senza error.
35	Sub, Function, Property non definita.
52	Nome o numero di file non valido.
53	File non trovato.
54	Modalità file non valida.
55	File già aperto.
57	Errore I/O di periferica.
58	File già esistente.
59	Lunghezza del record non valida.
61	Disco pieno.
62	Input oltre la fine del file.
63	Numero di record non valido.
67	Troppi file.
71	Disco non pronto.
74	Impossibile rinominare con unità diversa.
75	Errore di accesso al percorso/file.
76	Impossibile trovare il percorso

continua

Codice	Messaggio emesso
92	Ciclo For non inizializzato.
93	Stringa campione non valida.
31036	Errore di salvataggio su File.
31037	Errore di caricamento dal file.

Quello che segue è un esempio di come usare l'oggetto `Err`.

```
Private Sub Form_Load( )
  Dim X As Integer
  X = 0
  On Error GoTo mioErrore
  If X < 1 Or X > 9 Then
    Err.Raise vdObjectError + 111, , "Il numero è fuori
dall'intervallo"
  End If
  Debug.Print "OK!"
ResumeHere:
  Exit Sub
mioErrore:
  If Err.Number = vbObjecError + 111) Then
    Debug.Print Err.Description
  Else
    Debug.Print "Errore sconosciuto!"
  End If
  Resume resumeHere
End Sub
```

Questo esempio visualizza, nella finestra Immediata, la stringa `Il numero è fuori dall'intervallo`; la procedura `Form_Load( )` dichiara la variabile `X` di tipo `Integer` e le assegna il valore `0`. La routine quindi imposta un gestore di errore servendosi dell'enunciato `On Error`. A questo punto viene introdotto un enunciato `If` per determinare se il valore contenuto nella variabile `X` è al di fuori dell'intervallo tra 1 e 9. Poiché questa condizione risulta vera (la variabile `X` contiene il valore 0) la procedura utilizza il codice di errore personalizzato 111 (usando il valore `vbObjecError + 111`) e gli associa il messaggio `Il numero è fuori dall'intervallo`. Questo errore passa l'elaborazione a un gestore di errore posto dopo l'etichetta `mioErrore`. Il gestore si serve di un enunciato `If` per determinare se il numero di errore è pari a `vbObjectError + 111`; in caso affermativo, il gestore di errore visualizza nella finestra Immediata il messaggio servendosi della proprietà `Description` dell'oggetto `Err`. In caso contrario, il gestore visualizza il messaggio `Errore sconosciuto!`. Il gestore di errore esegue poi un enunciato

Resume che riporta l'esecuzione del programma all'etichetta ResumeHere. Questa etichetta appare immediatamente prima dell'enunciato Exit Sub. Dopo aver scritto questo codice, non verrà mai eseguito l'enunciato Debug.Print "OK!" poiché il codice stesso genera sempre un errore.

Enunciato On Error

Visual Basic consente alle routine di gestire gli errori di esecuzione; si ha anche la possibilità di definire la posizione della routine all'interno di una procedura servendosi, per questa operazione, dell'enunciato On Error.

La sintassi generale dell'enunciato On Error è la seguente.

```
On Error GoTo [etichetta ¦ lineaDiCodice]
```

La sintassi On Error GoTo lineaDiCodice consente alla routine per la gestione degli errori di avviare l'esecuzione a partire dall'argomento etichetta o lineaDiCodice. L'argomento etichetta o lineaDiCodice può corrispondere a una qualsiasi linea di codice contenente un'etichetta oppure al numero della linea di codice stessa. Se si verifica un errore di esecuzione, il controllo viene trasferito automaticamente all'etichetta o alla linea di codice indicata e viene avviato il gestore dell'errore localizzato in corrispondenza dell'etichetta o del numero di linea di codice. L'etichetta specificata o il numero di linea deve essere nella medesima procedura contenente l'enunciato On Error: in caso contrario, al momento della compilazione del codice sorgente, verrà emesso un messaggio di errore.

```
On Error Resume Next
```

La sintassi On Error Resume Next specifica che quando si verifica un errore di esecuzione il controllo verrà trasferito all'enunciato immediatamente successivo a quello che ha generato l'errore consentendo la continuazione dell'esecuzione dell'applicazione. Si consiglia di usare questa sintassi, in sostituzione di On Error GoTo, quando si accede direttamente agli oggetti.

Si utilizzi la sintassi On Error Resume Next in tutti quei casi in cui sia possibile ignorare l'errore che si è verificato. Per esempio, se, in un programma, si volesse eliminare un file di backup prima di crearne uno nuovo, si utilizzi l'enunciato On Error Resume Next. Se l'evento determinato dall'enunciato di eliminazione di un file di backup genera un errore (poiché il file di backup non esiste), il programma ignorerà semplicemente l'errore e procederà con la linea di codice successiva.

```
On Error GoTo 0
```

La sintassi `On Error GoTo 0` non fa altro che disabilitare qualsiasi gestore di errore presente nella procedura.

Si consiglia di scrivere il codice degli enunciati per la gestione degli errori alla fine della procedura; inoltre, prima dell'enunciato che gestisce il primo errore, si consiglia di inserire un enunciato `Exit Sub`, `Exit Function` o `Exit Property`.

Per maggiori informazioni sulla gestione degli errori, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello che segue è un esempio di come usare i vari enunciati per la gestione degli errori.

```
Private Sub Form_Load( )
  Dim X As Double
  Dim Y As Double
  Dim Z As Double

  On Error GoTo DivisionePerZero
  X = 1
  Y = 0
  Z = X / Y
  On Error GoTo 0
  Debug.Print X; "/"; Y; " ="; Z
  Exit Sub

DivisionePerZero:
  Debug.Print "Errore! Divisione per zero"
  Y = 2
  Resume
End Sub
```

Questo esempio genera la visualizzazione, nella finestra Immediata dei seguenti messaggi:

```
Errore! Divisione per zero
  1 / 2 = 0.5
```

La procedura `Form_Load( )` dichiara le variabili `X`, `Y` e `Z` di tipo `Double` e inizializza le prime due utilizzando, rispettivamente, i valori 1 e 0. La procedura imposta poi un gestore di errore servendosi dell'enunciato `On Error GoTo DivisionePerZero`. In seguito viene eseguita la divisione di `X` per `Y` e il risultato ottenuto viene memorizzato nella variabile `Z`. Questo enunciato genera un errore di divisione per zero, trasferendo il controllo all'etichetta `DivisionePerZero:`.

Il codice posto dopo l'etichetta `DivisionePerZero:` assegna il valore `2` alla variabile `Y` e, quindi, richiama l'enunciato `Resume`. Questo enunciato ripete l'esecuzione del comando che ha generato l'errore (`Z - X / Y`). Questa volta, poiché la variabile `Y` contiene il valore `2`, l'enunciato di assegnamento viene eseguito senza generare alcun tipo di errore. La procedura, quindi, disattiva il gestore di errore servendosi dell'enunciato `On Error GoTo 0` e visualizza i valori delle variabili all'interno della finestra **Immediata** uscendo infine dalla routine tramite l'esecuzione dell'enunciato `Exit Sub`. La presenza di quest'ultimo enunciato è necessaria per evitare che la procedura acceda di nuovo al gestore degli errori.

Enunciato If-Then Else

L'enunciato `If-Then Else` consente di prendere delle decisioni in base alla presenza di valori logici o di espressioni di tipo booleano.

Visual Basic prevede due sintassi principali per l'enunciato `If-Then Else` e la prima è la seguente.

```
If condizione Then [enunciati] [Else enunciatiElse]
```

In questa forma, l'enunciato `If-Then Else` inizia con la parola chiave `If` seguita immediatamente dopo dalle condizioni di verifica. La parola chiave `Then` viene posta alla fine delle condizioni da verificare e, di solito, viene seguita da uno o più enunciati (separati tra loro da due punti [:]). Il programma, se la condizione verificata risulta vera, esegue l'enunciato dopo la clausola `Then`. La clausola facoltativa `Else` può essere utilizzata per offrire uno o più enunciati che dovranno essere eseguiti quando le condizioni verificate risultano false.

La seconda forma dell'enunciato `If-Then Else` è la seguente.

```
If condizione Then
  [  enunciati]
  [  ElseIf condizione Then
    [condizioniElseIf] ...
  [  Else
    condizioniElse]]
End If
```

La sintassi utilizza blocchi di enunciati che sono più facilmente leggibili rispetto a quelli indicati nella prima sintassi; inoltre, questa sintassi prevede clausole `Else-If` il cui scopo è quello di verificare condizioni aggiuntive. Queste clausole trasformano l'enunciato `If-Then Else` in un enunciato che è in grado di prendere decisioni in base ad alternative multiple. In questo caso, il programma esamina

ogni condizione fino a quando non reperirà quella che risulta vera (ed esegue gli enunciati della clausola `Then` posti dopo la condizione vera) oppure esegue la clausola `Else` (che agisce come un elemento di raccolta di tutte le clausole).

Per maggiori informazioni circa l'utilizzo dell'enunciato `If` per prendere delle decisioni, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello che segue è un esempio pratico di un enunciato `If-Then Else`.

```
Private Sub Form_Load( )
  Dim X As Long
  X = 1000 * Rnd( ) + 1
  If X < 10 Then
    Debug.Print "1 cifra"
  ElseIf X < 100 Then
    Debug.Print "2 cifre"
  Else
    Debug.print "3 cifre"
  End If
End Sub
```

Questo esempio potrebbe visualizzare, nella finestra Immediata, il seguente risultato:

```
3 cifre
```

La procedura `Form_Load( )` dichiara la variabile locale `X` di tipo `Long` e le assegna un valore casuale, compreso tra 1 e 1000. La procedura utilizza un enunciato `If` per determinare di quante cifre è composto il valore della variabile `X`. La clausola `If` esamina la condizione `X < 10` e quindi visualizza, nella finestra Immediata, la stringa `1 cifra` solo nel caso in cui la condizione verificata risulti vera. La clausola `ElseIf` esamina la condizione `X < 100` e, se la condizione è vera, sempre nella finestra Immediata, visualizza la stringa `2 cifre`. La clausola finale `Else` visualizza nella finestra Immediata la stringa `3 cifre`. Poiché la possibilità di generare un numero casuale superiore a 99 è relativamente alta, questo codice, nella maggior parte dei casi, eseguirà l'enunciato associato alla clausola `Else` dopo aver esaminato il contenuto delle due condizioni precedenti.

Enunciato Resume

L'enunciato `Resume` indica al programma come recuperare l'esecuzione di un'elaborazione al termine di una routine per la gestione degli errori.

Vedi anche: in questa parte, la sezione che descrive l'oggetto `Err`.

La sintassi generale dell'enunciato `Resume` è la seguente.

Resume [0]

La sintassi `Resume` fa sì che l'esecuzione del programma venga ripresa dall'enunciato che ha generato l'errore, sempre che l'errore si sia verificato nella stessa procedura contenente il gestore di errore. Se l'errore si verifica in una procedura richiamata, l'esecuzione del programma verrà ripresa in corrispondenza dell'ultimo enunciato che ha richiamato la procedura che contiene la routine per la gestione degli errori. L'argomento `0` è facoltativo ed è un retaggio di GWBASIC e di BASIC che richiedevano l'indicazione di un numero di linea di codice. Il suo inserimento è stato determinato da una necessità di compatibilità con queste versioni precedenti.

Resume Next

La sintassi `Resume Next` fa sì che l'esecuzione del programma venga ripresa in corrispondenza dell'enunciato immediatamente successivo dell'enunciato che ha generato l'errore, sempre che l'errore si sia verificato nella stessa procedura contenente il gestore dell'errore. Se l'errore si è verificato in una procedura richiamata, l'esecuzione riprenderà con l'enunciato immediatamente successivo quello che ha chiamato l'ultima procedura contenente la routine per la gestione degli errori (oppure l'enunciato `On Error Resume Next`).

Resume numeroLinea

La sintassi `Resume numeroLinea` fa sì che l'esecuzione dell'applicazione venga ripresa in corrispondenza dell'etichetta o del numero della linea di codice e deve essere contenuta nella stessa procedura contenente il gestore di errore.

L'uso dell'enunciato `Resume` al di fuori di un gestore di errore causa un errore di esecuzione.

Il codice seguente è un esempio dell'uso delle tre forme dell'enunciato `Resume`.

```
Private Sub Form_Load( )
  Dim X As Double
  Dim Y As Double
  Dim Z As Double

  On Error GoTo DivisionePerZero
  X = 1
  Y = 0
  Z = X / Y
```

```
    On Error GoTo 0
    Debug.Print X; "/"; Y; " ="; Z

    On Error Resume next
    Z = X / 0
    Z = 1
    Debug.Print "Z ="; Z

    On Error GoTo Err2
    Z = X / 0
    ResumeErr2Here:
    Z = 10
    Debug.Print "Z ="; Z
    Exit Sub

    DivisionePerZero:
    Debug.Print "Errore! Divisione per zero"
    Y = 2
    Resume

    Err2:
    ResumeErr2Here
End Sub
```

Questo esempio genera il seguente risultato nella finestra Immediata.

```
Errore! Divisione per zero
 1/2 =0.5
Z = 1
z = 1
```

La procedura `Form_Load( )` contiene i seguenti enunciati per la generazione di errori e i seguenti enunciati `Resume`.

- La prima serie di enunciati che dividono la variabile `Y` (il cui valore è pari a `0`) richiama il gestore di errore dell'etichetta `DivisionePerZero:`. Questo gestore di errore assegna alla variabile `Y` il valore `2` e quindi esegue l'enunciato `Resume`. Questo enunciato indica al programma di riprendere l'esecuzione in corrispondenza della linea che ha generato l'errore.
- La seconda serie di enunciati che segue la divisione per 0 utilizza l'enunciato `On Error Resume Next` che indica la ripresa dell'esecuzione del programma in corrispondenza dell'enunciato `Z = 1`.
- La terza serie di enunciati che eseguono una divisione per `0` si serve dell'enunciato `On Error GoTo Err2` che fa sì che

l'esecuzione del programma venga ripresa in corrispondenza dell'etichetta Err2:.Questo gestore contiene l'enunciato ResumeErr2Here il cui scopo è quello di trasferire il controllo per l'esecuzione del programma in corrispondenza dell'etichetta ResumeErr2Here:. L'enunciato posto dopo l'etichetta assegna alla variabile Z il valore 10.

Enunciato Select Case

L'enunciato Select Case è un sistema molto versatile per prendere una decisione analizzando diverse alternative. Sebbene l'enunciato Select Case sia molto simile all'enunciato If-Else contenente clausole ElseIf, è molto più flessibile.

La sintassi generale dell'enunciato Select Case è la seguente.

```
Select Case verificaEspressione
  [  Case elencoEspressioni
    [enunciati]]...
  [  Case Else
    [enunciatiCaseElse]]
End Select
```

verificaEspressione corrisponde a un'espressione di tipo numerico o di tipo stringa. elencoEspressioni corrisponde a un elenco di valori separati da virgole indicati dopo la clausola Case. L'elenco dei valori contiene una o più delle forme seguenti.

```
espressione
espressione To espressione
[Is] operatoreConfronto espressione
```

La parola chiave To specifica un intervallo di valori e quando si utilizza questa parola chiave, prima di essa, dovrebbe comparire il valore minore. Si utilizza, invece, la parola chiave Is in associazione con operatori di confronto (con la sola eccezione di Is e Like) per specificare un intervallo di valori. Enunciati rappresenta uno o più enunciati eseguiti quando verificaEspressione trova una corrispondenza con una qualsiasi parte di elencoEspressioni. enunciatiCaseElse rappresenta uno o più enunciati eseguiti se verificaEspressione non trova una corrispondenza con una qualsiasi delle clausole Case.

Quando verificaEspressione trova una corrispondenza con una qualsiasi espressione Case elencoEspressioni, gli enunciati che seguono la clausola Case vengono eseguiti fino alla successiva clausola Case o, nel caso in cui si tratti dell'ultima clausola, fino a End Select.

Per maggiori informazioni sull'enunciato `Select Case`, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello che segue è un esempio di come usare l'enunciato `Select Case`.

```
Private Sub Form_Load( )
  Dim X As Long
  X = 1000 * Rnd( ) + 1
  Select Case X
    Case Is < 10
    Debug.Print "1 cifra"
  Case 10 To 99
    Debug.Print "2 cifre"
  Case Else
    Debug.Print "3 cifre"
  End Select
End Sub
```

Questo esempio visualizza, di solito, nella finestra Immediata, la seguente stringa:

```
3 cifre
```

La procedura `Form_Load( )` dichiara la variabile `X` di tipo `Long` e le assegna un valore casuale, compreso tra 1 e 1000. La procedura si serve dell'enunciato `Select Case` per determinare di quante cifre è composto il valore della variabile `X`. La clausola `Select Case` esamina il valore della variabile `X`. La prima clausola `Case` contiene `Is < 10` e, quando `X` sarà inferiore a 10, nella finestra Immediata verrà visualizzata la stringa `1 cifra`.

La seconda clausola `Case` contiene `10 To 99` e genera, sempre nella finestra Immediata la visualizzazione della stringa `2 cifre`. Infine, `Case Else` visualizza la stringa `3 cifre`. Poiché nella maggior parte dei casi la generazione di un numero casuale sarà superiore a 99, il codice eseguirà quasi sempre l'enunciato `Case Else` dopo aver esaminato le prime due condizioni e, quindi, visualizzerà nella finestra Immediata la stringa `3 cifre`.

Parte VII

Funzioni e Procedure

Le procedure e le funzioni si comportano come miniprogrammi inseriti in un programma più generale. Se per esempio si volessero acquistare 15 arance in un negozio che prevede di distribuirle solo in confezioni da 3, per raggiungere il risultato desiderato, si dovrebbe eseguire mentalmente il calcolo seguente: "5 confezioni da 3 arance danno come risultato 15 arance". Questo banale processo mentale non è altro che ciò che viene eseguito da una procedura o da una funzione all'interno del programma nel quale sono state inserite.

Argomenti trattati

- **Uso delle procedure per eseguire compiti nel contesto di programmi più generali**
- **Uscita dalle procedure**
- **Comprensione e uso di funzioni per calcolare e restituire valori all'interno di un programma**
- **Uscita dalle funzioni**

Enunciato Exit Function

L'enunciato `Exit Function` consente di uscire immediatamente dalla procedura `Function` corrispondente. L'esecuzione del programma riprende con l'enunciato immediatamente successivo quello che ha richiamato la procedura `Function`.

La sintassi generale dell'enunciato `Exit Function` è la seguente.

```
Exit Function
```

L'enunciato prevede due tipi fondamentali di utilizzo.

- ✦ Presenza in una procedura `Function` prima di un enunciato per la gestione degli errori della procedura stessa. Questo tipo di utilizzo evita che il programma esegua enunciati di gestione degli errori quando non se ne verifica alcuno.
- ✦ Inserimento all'interno di un enunciato `If` che esamina le condizioni per determinare se è necessario o meno uscire dalla procedura `Function`.

Quello che segue è un esempio di come sia possibile usare l'enunciato `Exit Function` all'interno di una procedura `Function` contenente elementi per la gestione degli errori.

```
Function DivideInts(X As Integer, Y As Integer) As Integer
  Dim Z As Integer
  On Error GoTo ZeroDivide ' attiva il gestore di errore
  Z = X / Y
  On Error GoTo 0 ' disattiva il gestore di errore
  DivideInts = Z
  ' esce dalla funzione
  Exit Function

' inizio del gestore di errore
ZeroDivide:
  Y = 1
  Resume
End Function
```

Questo esempio definisce la funzione `DivideInts( )` che restituisce il risultato della divisione dei valori delle variabili `X` e `Y` di tipo intero. La funzione attiva un gestore di errore prima di dividere questi argomenti e, quindi, lo disattiva dopo aver effettuato la divisione. La funzione imposta il valore restituito come il risultato della divisione e richiama quindi l'enunciato `Exit Function` per uscire dalla funzione. Gli enunciati che seguono `Exit Function` sono dei gestori di errore che iniziano immediatamente dopo l'etichetta `ZeroDivide:`.

Senza l'enunciato `Exit Function`, quando si richiama la funzione `DivideInts( )`, verrebbe presentato un errore di esecuzione.

Quello che segue è un esempio dell'uso di `Exit Function` all'interno di una procedura `Function` contenente un enunciato `If`.

```
Function DivideNums(X As Integer, Y As Integer) As Integer
  Dim Z As Integer
DivideNums = 0
  If Y = 0 Then Exit Function
  Z = X / Y
  DivideNums = Z
End Function
```

Questo esempio utilizza la funzione `DivideNums( )` che restituisce il risultato della divisione dei suoi argomenti `X` e `Y` di tipo intero. La funzione assegna il valore restituito di default `0` e quindi utilizza un enunciato `If` per determinare se il valore dell'argomento `Y` è pari a 0. Quando questa condizione risulta vera, il programma esegue l'enunciato `Exit Function` per uscire dalla funzione. In caso contrario la funzione divide gli argomenti, ne memorizza il risultato nella variabile `Z` e, quindi, utilizza il valore di tale variabile come valore restituito.

Enunciato Exit Sub

L'enunciato `Exit Sub` consente di uscire immediatamente dalla procedura `Sub` corrispondente. L'esecuzione del programma riprende con l'enunciato che viene immediatamente dopo l'enunciato che ha richiamato la procedura `Sub`.

La sintassi generale dell'enunciato `Exit Sub` è la seguente.

```
Exit Sub
```

Vi sono due usi generali dell'enunciato `Exit Sub`.

- L'enunciato `Exit Sub` compare in una procedura `Sub` prima di un enunciato per la gestione degli errori contenuto nella procedura stessa. Questo tipo di utilizzo evita che il programma esegua gli enunciati per la gestione degli errori quando non si verifica alcun errore.
- L'enunciato `Exit Sub` compare in un enunciato `If` che esamina una condizione per determinare se sia o meno necessario uscire da una procedura `Sub`.

Quello seguente è un esempio dell'uso dell'enunciato `Exit Sub` in una procedura `Sub` comprendente enunciati per la gestione degli errori.

```
Sub DivideInts(X As Integer, Y As Integer)
  Dim Z As Integer
  On Error GoTo ZeroDivide 'attiva il gestore di errore
  Z = X / Y
  On Error GoTo 0 ' disattiva il gestore di errore
  Debug.Print Z
  ' esce dalla procedura Sub
  Exit Sub

' qui inizia il gestore di errore
ZeroDivide:
  Y = 1
  Resume
End Sub
```

Questo esempio definisce la procedura `DivideInts( )` che divide fra di loro i valori dei propri argomenti interi `X` e `Y`. La procedura attiva un gestore di errore prima di dividere questi argomenti e, dopo aver eseguito la divisione, lo disattiva. La procedura visualizza il risultato della divisione e poi richiama un enunciato `Exit Sub` per uscire dalla procedura. Gli enunciati successivi a `Exit Sub` sono dei gestori di errore che iniziano immediatamente dopo l'etichetta `ZeroDivide:`.

La mancanza dell'enunciato `Exit Sub` genera, in fase di esecuzione, quando viene richiamata la procedura `DivideInts( )`, un errore.

L'esempio seguente vuole dimostrare l'uso dell'enunciato `Exit Sub` in una procedura `Sub` contenente un enunciato `If`.

```
Sub DivideNums(X As Integer, Y As Integer)
  Dim Z As Integer
  If Y = 0 Then Exit Sub
  Z = X / Y
  Debug Print Z
End Sub
```

L'esempio utilizza una procedura `DivideNums( )` che divide i valori interi dei propri argomenti `X` e `Y`. La procedura utilizza un enunciato `If` per determinare se il valore dell'argomento `Y` sia pari a zero. Quando questa condizione è vera, il programma esegue l'enunciato `Exit Sub` per uscire dalla procedura. In caso contrario, la procedura divide gli argomenti, ne memorizza il risultato nella variabile locale `Z` e visualizza il valore ottenuto in tale variabile.

Funzioni

Per eseguire alcune operazione e restituire un valore,Visual Basic si serve di funzioni.

La sintassi generale di una funzione è la seguente.

```
[Public ¦ Private ¦ Friend] [Static] Function nome
[(elencoargomenti)] [As tipo]
[enunciati]
[nome = espressione]
[Exit Function]
[enunciati]
[nome = espressione]
Exit Function
```

La dichiarazione di una funzione semplice inizia con la parola chiave `Function` e deve concludersi con un enunciato che contenga comunque le parole `End Function`.La tabella seguente schematizza le parti facoltative che possono apparire prima della parola chiave `Function`.

Parte	*Descrizione*
`Public`	Indica che la funzione sarà accessibile a tutte le altre procedure di tutti i moduli. Se `Public` si utilizza in un modulo contenente `Option Private`, la funzione *non* sarà disponibile dall'esterno del progetto.
`Private`	Indica che la funzione sarà accessibile solo alle altre procedure del modulo nella quale la funzione è dichiarata.
`Friend`	Utilizzato solo in un modulo classe. Indica che la funzione sarà visibile dall'interno del progetto ma non dal controllore di un'istanza di un oggetto.
`Static`	Indica che le variabili locali della funzione vengono conservate fra una chiamata e l'altra. L'attributo `Static` non influisce sulle variabili che vengono dichiarate al di fuori della funzione, anche nel caso in cui queste vengano utilizzate all'interno della funzione stessa.

La voce successiva `nome` corrisponde al nome della funzione e deve attenersi alle convenzioni standard per l'assegnazione dei nomi in Visual Basic.

`elencoargomenti` è facoltativo (sebbene spesso utilizzato) e rappresenta un elenco di argomenti tra di loro separati da virgole e che viene passato alla funzione quando questa verrà chiamata.

L'argomento `elencoargomenti` prevede la seguente sintassi.

`[Optional] [ByVal ¦ ByRef] [ParamArray] nomeVar[( )] [As tipo] [= valoreDefault]`

Il termine *argomento*, in Visual Basic, viene spesso indicato, da altri linguaggi di programmazione, con il termine *parametro* e, nel contempo, si servono del termine *argomento* per fare riferimento al *valore* fornito al parametro quando si verifica una chiamata di funzione.

La tabella seguente riporta le parti per l'indicazione di un argomento.

Parte	*Descrizione*
`Optional`	Indica che un argomento non viene espressamente richiesto. Se utilizzato, tutti gli argomenti successivi contenuti in `elencoargomenti`, debbono necessariamente essere considerati facoltativi e dichiarati utilizzando la parola chiave `Optional`. La parola chiave `Optional` non deve essere utilizzata per alcun argomento nel caso in cui venga utilizzato anche `ParamArray`.
`ByVal`	Indica che l'argomento viene passato in base al proprio valore.
`ByRef`	Indica che l'argomento viene passato in base al proprio riferimento (default).
`ParamArray`	Utilizzato solo come ultimo argomento di `elencoargomenti` per indicare che l'argomento finale è una matrice di tipo `Optional` di elementi `Variant`. La parola chiave `ParamArray` consente di fornire un numero arbitrario di argomenti e non deve essere usato in presenza di `ByVal` `ByRef` od `Optional`.
`NomeVar`	Corrisponde al nome della variabile che rappresenta l'argomento e deve seguire le convenzioni standard per l'assegnazione dei nomi in Visual Basic.
`Tipo`	Definisce il tipo di dati dell'argomento passato alla procedura; può essere `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (solo di lunghezza variabile), `Object` o `Variant`.
`valoreDefault`	Una qualsiasi costante o espressione costante. Questo elemento è valido per i parametri `Optional` e se il tipo è `Object`, sarà possibile solo assegnare il valore di default `Nothing`.

Dopo `elencoargomenti` della sintassi generale della funzione vi è `tipo` che definisce, facoltativamente, il tipo di dati del valore restituito dalla funzione. Sono ammissibili i tipi `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (solo a lunghezza variabile), `Object` e `Variant`, o qualsiasi altro tipo di dati personale. Una funzione non può restituire una matrice ma può restituire un valore `Variant` contenente una matrice. Per restituire un valore, gli enunciati della funzione devono contenere un enunciato che assegni un valore al nome della funzione. Se non si assegna un valore a `nome`, la funzione genera un valore di default: una funzione numerica restituirà 0, una funzione stringa una stringa vuota (o di lunghezza zero, [`""`]) e una funzione di tipo `Variant` restituirà `Empty`. Una funzione che restituisce un riferimento a un oggetto restituirà `Nothing` nel caso in cui a `nome` (tramite `Set`) non venga assegnato alcun riferimento a oggetto.

L'enunciato `Exit Function` determina l'uscita prematura dalla funzione e in una funzione possono apparire più enunciati `Exit Function`.

Per maggiori informazioni sulle funzioni, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Gli esempi seguenti hanno lo scopo di dimostrare l'uso delle funzioni: il primo restituisce un valore di tipo numerico mentre il secondo restituisce un tipo stringa.

```
Public Function Power(X As Double, Optional Y = 2) As
Double
  If (X < > 0) Then
    Power = X ^ 1
  Else
    Power = -1E+300
  End If
End Function
Public Function ShowPow(X As Double, Optional Y As Double =
2) As String
  ShowPow = "(" + CStr(X) + ")" ^ (" + Cstr(Y) + ") = " +
CStr(Power(X, Y))
End Function
Private Sub Form_Load( )
  Dim A As Double
  Dim B As Double
  A = 5
  B = 3
  Debug.Print ShowPow(A)
  Debug.Print ShowPow(A, B)
  Debug.Print ShowPow(-A, B)
  Debug.Print ShowPow(-A, -B)
End Sub
```

Questo esempio genera la visualizzazione, nella finestra Immediata, del seguente risultato.

```
(5)^(2) = 25
(5)^(3) = 125
(-5)^(3) = -125
(-5)^(-3) = -0.008
```

L'esempio contiene le funzioni `Power( )` e `ShowPow( )`. La prima restituisce il risultato ottenuto dall'elevazione di un numero a una potenza. La funzione genera un risultato di tipo `Double` ed è associata a due argomenti (`X` e `Y`) anch'essi di tipo `Double`. Il secondo argomento è facoltativo e gli viene assegnato il valore di default 2. Pertanto, quando si omette di inserire il valore per tale argomento, la funzione restituisce automaticamente l'elevazione al quadrato dell'argomento `X`. La funzione `Power( )` fa uso di un enunciato `If` per esaminare se il valore dell'argomento `X` non sia pari a 0. Quando questa condizione è vera, la funzione restituisce il valore di `X` elevato alla potenza corrispondente al valore di `Y`. In caso contrario, la funzione utilizza il valore `-1E+300` (corrispondente a un numero negativo molto grande che riflette un codice numerico di errore).

La funzione `ShowPow( )` restituisce, invece, una stringa che formatta il valore degli argomenti `X` e `Y` e genera il risultato di `X` elevato a `Y`. La funzione prevede un risultato di tipo `String` e due argomenti (`X` e `Y`) di tipo `Double`. Il secondo argomento, `Y`, è facoltativo e contiene il valore di default 2. Pertanto, quando non si assegna un valore a questo argomento, la funzione visualizza l'elevazione al quadrato dell'argomento `X`. La funzione `ShowPow( )` richiama la funzione `Pow()` e passa gli argomenti `X` e `Y` come valori agli argomenti della funzione `Power( )`.

La procedura `Form_Load( )` dichiara le variabili locali `A` e `B` di tipo `Double` assegnando loro, rispettivamente, i valori 5 e 3. Successivamente la procedura richiama per quattro volte la funzione `ShowPow()` tramite quattro enunciati `Print` consecutivi. La prima chiamata di funzione passa il valore dell'argomento `A` alla funzione `ShowPow( )`. Questa chiamata fa sì che la funzione `ShowPow( )` utilizzi il valore di default (2) per l'argomento `Y`. La seconda chiamata passa le variabili `A` e `B` come valori di argomento alla funzione `ShowPow( )`. La terza chiamata passa le variabili `-A` e `B` come valori di argomento per la funzione `ShowPow( )`. L'ultima chiamata, infine, passa le variabili `-A` e `-B` come valori di argomento per la funzione `ShowPow( )`.

Procedure

Una procedura è un componente di un'applicazione che esegue una o più operazioni all'interno del contesto di un programma di dimensioni più estese. Si può spesso pensare a una procedura come a un miniprogramma contenuto in un programma di più ampio respiro. Si chiama una procedura e si passano alla stessa valori per i suoi argomenti, in modo che tali valori siano in grado di fornire i dati richiesti. In questo modo la procedura sarà in grado di eseguire i compiti alla stessa richiesti.

La sintassi generale per una procedura è la seguente.

```
[Private ¦ Public ¦ Friend] [Static] Sub nome
[(elencoargomenti)]
  [  enunciati]
  [  Exit Sub]
  [  enunciati]
End Sub
```

La dichiarazione di una procedura semplice inizia con la parola chiave `Sub` e termina con le parole chiavi `End Sub`.

Nella tabella seguente vengono riportati gli elementi facoltativi che possono precedere la parola chiave `Sub`.

Parte	*Descrizione*
`Public`	Indica che la procedura sarà accessibile a tutte le altre procedure di tutti i moduli. Se `Public` si utilizza in un modulo contenente `Option Private`, la procedura *non* sarà disponibile dall'esterno del progetto.
`Private`	Indica che la procedura sarà accessibile solo alle altre procedure del modulo da cui la procedura è dichiarata.
`Friend`	Utilizzato solo in un modulo classe. Indica che la procedura sarà visibile dall'interno del progetto ma non dal controllore di un'istanza di un oggetto.
`Static`	Indica che le variabili locali della procedura vengono conservate fra una chiamata e l'altra. L'attributo `Static` non influisce sulle variabili che vengono dichiarate al di fuori della procedura, anche nel caso in cui queste vengano utilizzate all'interno della procedura stessa.

La voce successiva `nome` corrisponde al nome della procedura e deve attenersi alle convenzioni standard per l'assegnazione dei nomi in Visual Basic.

`elencoargomenti` è facoltativo (sebbene spesso utilizzato) e rappresenta un elenco di argomenti tra di loro separati da virgole e che viene passato alla procedura quando questa verrà chiamata.

L'argomento `elencoargomenti` prevede la seguente sintassi.

```
[Optional] [ByVal ¦ ByRef] [ParamArray] nomeVar[( )] [As
tipo] [= valoreDefault]
```

Il termine *argomento*, in Visual Basic, viene spesso indicato, da altri linguaggi di programmazione, con il termine *parametro* e, nel contempo, si servono del termine *argomento* per fare riferimento al *valore* fornito al parametro quando si verifica una chiamata di procedura.

La tabella seguente riporta le parti per l'indicazione di un argomento.

Parte	Descrizione
`Optional`	Indica che un argomento non viene espressamente richiesto. Se utilizzato, tutti gli argomenti successivi contenuti in `elencoargomenti`, debbono necessariamente essere considerati facoltativi e dichiarati utilizzando la parola chiave `Optional`. La parola chiave `Optional` non deve essere utilizzata per alcun argomento nel caso in cui venga utilizzato anche `ParamArray`.
`ByVal`	Indica che l'argomento viene passato in base al proprio valore. Questo tipo di argomento passa una copia del valore dell'argomento (che può corrispondere a una variabile o a un'espressione) alla procedura. Ogni modifica apportata a tale argomento all'interno della procedura non influisce sul valore originario dell'argomento.
`ByRef`	Indica che l'argomento viene passato in base al proprio riferimento (condizione di default). Questo tipo di argomento passa un riferimento del valore dell'argomento (che può corrispondere a una variabile) alla procedura. Qualsiasi modifica apportata all'argomento della procedura influisce sul valore originale dell'argomento.
`ParamArray`	Utilizzato solo come ultimo argomento di `elencoargomenti` per indicare che l'argomento finale è una matrice di tipo `Optional` di elementi `Variant`. La parola chiave `ParamArray` consente di fornire un numero arbitrario di argomenti e non deve essere usata in presenza di `ByVal` `ByRef` od `Optional`.

(continua)

Parte	Descrizione
NomeVar	Corrisponde al nome della variabile che rappresenta l'argomento e deve seguire le convenzioni standard per l'assegnazione dei nomi in Visual Basic.
Tipo	Definisce il tipo di dati dell'argomento passato alla procedura; può essere Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String (solo di lunghezza variabile), Object o Variant.
valoreDefault	Una qualsiasi costante o espressione costante. Questo elemento è valido per i parametri Optional e se il tipo è Object, sarà possibile solo assegnare il valore di default Nothing.

L'enunciato `Exit Sub` determina un'uscita anticipata dalla procedura che, tra l'altro, può prevedere la presenza di più enunciati di questo tipo.

Vedi anche: in questa parte, la sezione che descrive l'enunciato "Exit Sub".

Per richiamare una procedura si deve utilizzare l'enunciato

```
Call nomeProcedura([valoriElencoArgomenti])
```

Si ha anche la possibilità di omettere la parola chiave `Call` e le parentesi tonde che racchiudono i valori dell'elenco degli argomenti e, in questo caso, la sintassi generale per richiamare una procedura è la seguente.

```
nomeProcedura [ValoriElencoArgomenti]
```

Per maggiori informazioni sulle procedure, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello seguente è un esempio di come utilizzare una procedura.

```
Public Function Power(X As Double, Optional Y = 2) As
Double
  If (X < > 0) Then
    Power = X ^ Y
  Else
    Power = -1E+300
  End If
End Function
Public Sub ShowPow(X As Double, Optional Y As Double = 2)
  Debug.Print "(" + CStr(X) + ")"^(" + CStr(Y) + ") ? " +
CStr(Power (X, Y))
End Sub
```

```
Private Sub Form_Load( )
  Dim A As Double
  Dim B As Double
  A = 5
  B = 3
  ShowPow A
  Call ShowPow(A, B) ' uso della parola chiave Call
  ShowPow -A, B
  ShowPow -A, -B
End Sub
```

Questo esempio visualizza, nella finestra Immediata i seguenti risultati.

```
(5)^(2) = 25
(5)^(3) = 125
(-5)^(3) = -125
(-5)^(-3) = -0.008
```

L'esempio contiene le funzioni `Power( )` e `ShowPow( )`. La prima restituisce il risultato ottenuto dall'elevazione di un numero a una potenza. La funzione genera un risultato di tipo `Double` ed è associata a due argomenti (`X` e `Y`) anch'essi di tipo `Double`. Il secondo argomento è facoltativo e gli viene assegnato il valore di default 2. Pertanto, quando si omette di inserire il valore per tale argomento, la funzione restituisce automaticamente l'elevazione al quadrato dell'argomento `X`.

La funzione `Power( )` fa uso di un enunciato `If` per esaminare se il valore dell'argomento `X` non sia pari a 0. Quando questa condizione è vera, la funzione restituisce il valore di `X` elevato alla potenza corrispondente al valore di `Y`. In caso contrario, la funzione utilizza il valore `-1E+300` (corrispondente a un numero negativo molto grande che riflette un codice numerico di errore).

La procedura `ShowPow( )` restituisce, invece, una stringa che formatta il valore degli argomenti `X` e `Y` e genera il risultato di `X` elevato a `Y`. La procedura prevede due argomenti (`X` e `Y`) di tipo `Double` dove il secondo è facoltativo e prevede l'assegnazione del valore di default 2. Pertanto, quando non si assegna un valore a questo argomento, la funzione visualizza l'elevazione al quadrato dell'argomento `X`.

La procedura `ShowPow( )` richiama la funzione `Pow( )` e passa i valori degli argomenti `X` e `Y` agli argomenti della funzione `Power( )`.

La procedura `Form_Load( )` dichiara le variabili locali `A` e `B` di tipo `Double` assegnando loro, rispettivamente, i valori 5 e 3. Successivamente, la procedura richiama per quattro volte la funzione `ShowPow( )` tramite quattro enunciati `Print` consecutivi.

La prima chiamata di funzione passa il valore dell'argomento A alla funzione ShowPow( ). Questa chiamata fa sì che la funzione ShowPow( ) utilizzi il valore di default 2 per l'argomento Y. La seconda chiamata passa le variabili A e B come valori di argomento alla funzione ShowPow( ). La terza chiamata passa le variabili -A e B come valori di argomento per la funzione ShowPow( ). L'ultima chiamata, infine, passa le variabili -A e -B come valori di argomento per la funzione ShowPow( ).

Parte VIII

Manipolazione delle stringhe

Le stringhe, nell'accezione più estesa, non sono altro che testo. Questa parte si occuperà di tutte quelle operazioni che si possono avviare sulle stringhe come, per esempio, la loro conversione in altri tipi di dati, l'isolamento di frammenti di stringa, la loro espansione, riduzione, modifica eccetera.

Argomenti trattati

Concatenazione di stringhe

Visual Basic dà la possibilità di utilizzare l'operatore "+" per *concatenare* (o collegare), in un solo enunciato, due o più stringhe. La sintassi generale per l'uso dell'operatore "+" è la seguente.

```
espressione1 + espressione2 + espressione3 + ...
```

`espressione 1`,`espressione2` ed `espressione3` corrispondono a espressioni di tipo stringa che contengono variabili stringa, elementi testuali (racchiusi tra doppi apici [`" "`]) o una loro combinazione.

Quello che segue è un esempio dell'uso dell'operatore "+" per concatenare varie stringhe.

```
Private Sub Form_Load( )
  Dim Str1 As String
  Dim Str2 As String
  Dim Str3 As String
  Str1 = "Ciao"
  Str2 = "mondo"
  Str3 = Str1 + " " + Str2 + "!"
  Debug.Print Str3
End Sub
```

Il risultato di questo codice visualizza, nella finestra Immediata, il seguente risultato.

```
Ciao mondo!
```

La procedura `Form_Load( )` dichiara le variabili stringa locali `Str1`, `Str2` e `Str3` assegnando, alle prime due, rispettivamente, i valori `"Ciao"` e `"mondo"`. La procedura quindi concatena queste due variabili stringa con altri elementi di tipo testuale e assegna il risultato finale alla variabile `Str3`. Infine, la procedura visualizza, nella finestra Immediata, il valore della variabile `Str3` servendosi, per questa operazione, di un enunciato `Print`.

Conversione tra stringhe e altri tipi di dati

Visual Basic prevede la presenza del tipo di dati `Variant` il cui scopo è quello di eseguire conversioni automatiche fra i dati stringa e i dati di tipo numerico. Il tipo di dati `Variant` si comporta come una stringa all'interno di un'espressione stringa, come un numero in un'espressione numerica, come un operatore booleano in un'espressione di tipo booleano, e così via.

Vedi anche: nella *Parte III*, la sezione che descrive le classi e i tipi di dati.

La tabella seguente riporta le varie funzioni che si potranno usare per convertire i tipi di dati.

Funzione	*Tipo restituito*	*Intervallo per gli argomenti*
`CBool`	Booleano	Una qualsiasi espressione stringa o numerica.
`CByte`	Byte	Da 0 a 255.
`CCur`	Currency	Da –922.337.203.685. 477,5808 a –922.377. 203.685.477,5807.
`CDate`	Date	Una qualsiasi espressione di tipo `Date`.
`CDbl`	Double	+/–79.228.162.514. 254.337.393.543. 950.335 per i numeri senza cifre decimali, oppure +/–7,9228162514243 37393543950335 per i numeri con 28 cifre decimali. Il più basso valore diverso da zero è pari a 0,00000000000 00000000000000001.
`CInt`	Integer	Da –32.768 a 32.767; i valori frazionari vengono automaticamente arrotondati.
`CLng`	Long	Da –2.147.483.647 a 2.147.483.647; le frazioni vengono automaticamente arrotondate.
`CSng`	Single	Da –3,402823E38 a –1,401298E-45 per i valori negativi; da 1,401 298E-45 a 3,402823E38 per i valori positivi.
`CVar`	Variant	Lo stesso intervallo del tipo `Double` per i valori numerici e `String` per quelli di tipo stringa.
`CStr`	String	Il valore restituito da `CStr` dipende dall'argomento dell'espressione. Per esempio, se l'espressione è di tipo booleano, la funzione restituirà `True` oppure `False`. Se l'espressione è una data, la funzione restituirà un valore di tipo `Date`.

L'esempio che segue dimostra come sia possibile utilizzare alcune funzioni di conversione dei tipi di dati.

```
Private Sub Form_Load( )
  Dim miaStr As String
  Dim Ok As Boolean
  Dim mioDbl As Double
  Dim mioLng As Long
  ' verifica CBool
  miaStr = "true"
  Ok = CBool(miaStr)
  Debug.Print "Ok è "; Ok
  ' verifica CDbl
  miaStr = "123.45"
  mioDbl = CDbl(miaStr)
  Debug.Print "mioDbl è "; mioDbl
  ' verifica CLng
  miaStr = "1234567890"
  mioLng = Clng(miaStr)
  Debug.Print "mioLng è "; mioLng
End Sub
```

L'esempio precedente visualizza, nella finestra **Immediata**, il seguente risultato.

```
Ok è True
MioDbl è 123.45
MioLng è 1234567890
```

La procedura `Form_Load( )` dichiara le variabili seguenti.

- `miaStr` come `String`.
- `Ok` come `Double`.
- `mioDbl` come `Boolean`.
- `mioLng` come `Long`.

La procedura `Form_Load( )`, poi, esegue le seguenti operazioni.

- Assegna alla variabile `miaStr` la stringa letterale `"true"`.
- Converte il contenuto della variabile `miaStr` in un risultato di tipo booleano servendosi della funzione `CBool( )` e assegna il risultato alla variabile `Ok`.
- Visualizza il valore della variabile `Ok` nella finestra **Immediata** servendosi, per questa operazione, dell'enunciato `Print`.
- Assegna alla variabile `miaStr` la stringa letterale `"123.45"`.
- Converte il contenuto della variabile `miaStr` in un risultato di tipo `Double` servendosi della funzione `CDbl( )` e assegna il risultato finale alla variabile `mioDbl`.

- Visualizza il valore della variabile mioDbl nella finestra Immediata servendosi di un enunciato Print.
- Assegna alla variabile miaStr il valore letterale "1234567890".
- Converte il contenuto della variabile miaStr in un risultato di tipo Long servendosi della funzione CLng( ) e assegna il risultato finale alla variabile mioLng.
- Visualizza il valore della variabile nioLng nella finestra Immediata tramite l'enunciato Print.

InStr ( )

La funzione di Visual Basic InStr( ) permette di ricercare una stringa all'interno di una stringa di dimensioni più estese.

Vedi anche: in questa parte, la sezione che descrive la funzione InstrRev( ).

La sintassi generale della funzione InStr( ) è la seguente.

```
InStr([inizio, ]stringa1, stringa2[, confronto])
```

L'argomento facoltativo inizio corrisponde a un'espressione numerica che imposta la posizione iniziale di ogni ricerca. Quando si omette il valore per questo argomento, la ricerca inizia in corrispondenza del primo carattere. Se inizio contiene Null (la parola chiave Null viene usata come sottotipo del tipo Variant per indicare che una variabile Variant non contiene dati validi), viene emesso un errore. L'argomento inizio viene necessariamente richiesto nel caso in cui si utilizzi anche l'argomento confronto. L'argomento stringa1 corrisponde alla stringa di ricerca mentre l'argomento confronto specifica il tipo di confronto da avviare sulla stringa. L'argomento confronto può essere anche omesso, oppure potrà essere pari a -1, 0, 1 o 2. Il valore 0 (assunto per default) esegue un confronto di tipo binario; il valore 1 un confronto testuale (il valore -1 esegue un confronto utilizzando l'impostazione dell'istruzione Option Compare); il valore 2 (utilizzato solo per Microsoft Access) esegue un confronto in base alle informazioni contenute nel database. La funzione restituisce 0 nel caso in cui non possa reperire la stringa ricercata.

Per maggiori informazioni sulla ricerca di stringhe, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Ecco un esempio dell'uso della funzione InStr( ).

```
Private Sub Form_Load( )
  Dim StringaPrincipale As String
```

```
  Dim StringaCercata As String
  Dim Indice As Integer

  ' assegna le stringhe
  StringaPrincipale = "123123123"
  StringaCercata = "23"
  Indice = Instr(StringaPrincipale, StringaCercata)
  Do While Indice > 0
    Debug.Print "Trovato alla posizione "; Indice
    Indice = InStr(Indice + 1, StringaPrincipale,
StringaCercata)
  Loop
End Sub
```

Questo esempio visualizza, nella finestra Immediata, le seguenti linee:

```
Trovato alla posizione 2
Trovato alla posizione 5
Trovato alla posizione 8
```

La procedura `Form_Load( )` dichiara la variabile `StringaPrincipale` di tipo `String` (nella quale viene memorizzata la stringa principale) e la variabile `StringaCercata` (nella quale viene memorizzata la stringa di ricerca) anch'essa di tipo `String`. Viene anche dichiarata la variabile `Indice` di tipo `Integer`. La procedura assegna le stringhe letterali `"123123123"` e `"23"`, rispettivamente, alle variabili `StringaPrincipale` e `StringaCercata`. La procedura quindi reperisce nella stringa principale la prima concordanza della stringa di ricerca servendosi, per questa operazione, della funzione `InStr( )`. I valori dell'argomento per questa chiamata corrispondono alle variabili `StringaPrincipale` e `StringaCercata`.

La chiamata alla funzione `InStr( )` ricerca i caratteri all'interno della variabile `StringaPrincipale` a partire dalla prima posizione della stringa principale e ne memorizza il risultato nella variabile Indice. Per visualizzare il risultato di una ricerca terminata con esito positivo, e per procedere con la ricerca di ulteriori concordanze, la procedura si serve di un ciclo `Do While`.

Il ciclo `Do While` avvia una serie di iterazioni fintanto che la variabile `Indice` non conterrà un valore positivo. Il ciclo `Do While` si serve di due enunciati: il primo visualizza il valore della variabile `Indice` mentre il secondo ricerca la concordanza successiva richiamando, per questa operazione, la funzione `InStr( )`.

I valori dell'argomento di questa chiamata sono `Indice + 1`, `StringaPrincipale` e `StringaCercata`. Il primo valore di argomento garantisce che la ricerca venga rivolta verso la concordanza successiva all'interno della stringa principale.

InstrRev( )

La funzione di `InstrRev( )` è molto simile alla `InStr( )`, eccetto per il fatto che la ricerca viene effettuata al contrario, ossia dall'ultimo carattere fino al primo.

Vedi anche: in questa parte, la sezione che descrive la funzione `InStr ( )`.

La sintassi generale della funzione `InstrRev( )` è la seguente.

```
InstrRev(stringa 1, stringa 2 [,inizio] [,confronto])
```

L'argomento facoltativo `inizio` corrisponde a un'espressione numerica che imposta la posizione iniziale (in riferimento all'ultimo carattere della stringa) di ogni ricerca. Per esempio, il valore `4` per questo argomento indica che la ricerca deve partire 4 caratteri prima della fine della stringa. Quando si omette il valore per questo argomento, la ricerca inizia in corrispondenza dell'ultimo carattere. Se `inizio` contiene `Null` (la parola chiave `Null` viene usata come sottotipo del tipo `Variant` per indicare che una variabile `Variant` non contiene dati validi), viene emesso un errore. L'argomento `inizio` viene necessariamente richiesto nel caso in cui si utilizzi anche l'argomento `confronto`. L'argomento `stringa1` corrisponde alla stringa di ricerca mentre l'argomento `confronto` specifica il tipo di confronto da avviare sulla stringa. L'argomento `confronto` può essere anche omesso, oppure potrà essere pari a `-1`,`0`, `1`,`2`. Il valore `0` (assunto per default) esegue un confronto di tipo binario; il valore `1` un confronto testuale (Il valore `-1` esegue un confronto utilizzando l'impostazione dell'istruzione Option Compare); il valore `2` (utilizzato solo per Microsoft Access) esegue un confronto in base alle informazioni contenute nel database. La funzione restituisce 0 nel caso in cui non possa reperire la stringa ricercata.

Per maggiori informazioni sulla ricerca di stringhe, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Ecco un esempio dell'uso della funzione `InstrRev( )`.

```
Private Sub Form_Load( )
  Dim StringaPrincipale As String
  Dim StringaCercata As String
  Dim Indice As Integer
  ' assegna le stringhe
  StringaPrincipale = "123123123"
  StringaCercata = "23"
  Indice = InstrRev(StringaPrincipale, StringaCercata)
```

```
  Do While Indice > 0
    Debug.Print "Trovato alla posizione "; Indice
    Indice = InstrRev(Left(StringaPrincipale, Indice),
StringaCercata)
  Loop
End Sub
```

Questo esempio visualizza, nella finestra Immediata, le seguenti linee:

```
Trovato alla posizione 8
Trovato alla posizione 5
Trovato alla posizione 2
```

La procedura `Form_Load( )` dichiara la variabile `StringaPrincipale` di tipo `String` (nella quale viene memorizzata la stringa principale) e la variabile `StringaCercata` (nella quale viene memorizzata la stringa di ricerca) anch'essa di tipo `String`. Viene anche dichiarata la variabile `Indice` di tipo `Integer`. La procedura assegna le stringhe letterali `"123123123"` e `"23"`, rispettivamente, alle variabili `StringaPrincipale` e `StringaCercata`. La procedura quindi reperisce nella stringa principale l'ultima concordanza della stringa di ricerca servendosi, per questa operazione, della funzione `InstrRev( )`. I valori dell'argomento per questa chiamata corrispondono alle variabili `StringaPrincipale` e `StringaCercata`.

La chiamata alla funzione `InstrRev( )` ricerca i caratteri all'interno della variabile `StringaPrincipale` a partire dall'ultima posizione della stringa principale e ne memorizza il risultato nella variabile `Indice`. Per visualizzare il risultato di una ricerca terminata con esito positivo, e per procedere con la ricerca di ulteriori concordanze, la procedura si serve di un ciclo `Do While`.

Il ciclo `Do While` avvia una serie di iterazioni fintanto che la variabile `Indice` non conterrà un valore positivo. Il ciclo `Do While` si serve di due enunciati: il primo visualizza il valore della variabile `Indice` mentre il secondo ricerca la concordanza precedente richiamando, per questa operazione, la funzione `InstrRev( )`. I valori dell'argomento di questa chiamata sono `Left (StringaPrincipale, Indice)` e `StringaCercata`. Il primo valore di argomento garantisce che la ricerca venga rivolta verso la concordanza precedente all'interno della stringa principale rimanente (ossia quella che si trova a sinistra del valore della variabile `Indice`).

Left( )

La funzione `Left( )` permette di estrarre alcuni caratteri da una stringa.

Vedi anche: in questa parte, le sezioni che descrivono le funzioni `Right( )` e `Mid( )`.

La sintassi generale per la funzione `Left( )` è la seguente.

```
Left(stringa, lunghezza)
```

Il parametro `stringa` corrisponde a un'espressione di tipo `String` che contiene i caratteri da estrarre. Se il valore di questa espressione stringa è pari a `Null`, la funzione estrae `Null` (che corrisponde alla parola chiave utilizzata come sottotipo di `Variant` per indicare che una variabile `Variant` non contiene dati validi). Il parametro `lunghezza`, invece, specifica il numero di caratteri da estrarre. Se l'argomento di questo parametro è pari a 0, la funzione restituisce una stringa vuota (`""`) e se l'argomento di questo parametro è pari o superiore al numero dei caratteri contenuti nella stringa, la funzione estrarrà tutti i caratteri.

Quello seguente è un esempio di come usare la funzione `Left( )`.

```
Private Sub Form_Load( )
  Dim miaStr As String
  miaStr = "1234567890"
  miaStr = Left(miaStr, 3)
  Debug.Print miaStr
End Sub
```

Questo esempio visualizza, nella finestra Immediata, il seguente risultato:

```
123
```

La procedura `Form-Load( )` dichiara la variabile stringa locale `miaStr` e le assegna la stringa letterale `"1234567890"`. La procedura quindi estrae dalla variabile `miaStr` i primi tre caratteri servendosi, per questa operazione, della funzione `Left( )`. Il risultato di questa estrazione viene memorizzato di nuovo all'interno della stessa variabile. La chiamata della funzione `Left( )` prevede gli argomenti `miaStr` e `3` che specificano, rispettivamente, la stringa dalla quale eseguire l'estrazione e il carattere dal quale iniziare l'estrazione.

Nota. La funzione `Left(stringa, lunghezza)` si comporta nello stesso modo della funzione `Mid(stringa, 1, lunghezza)`.

Len ( )

La funzione `Len( )` restituisce il numero dei caratteri di una stringa.

La sintassi generale della funzione `Len( )` è la seguente.

```
Len(stringa)
```

Questa funzione restituisce il numero dei caratteri del parametro specificato.

Ecco un esempio dell'uso della funzione.

```
Private Sub Form_Load( )
  Dim miaStringa As String
  miaStringa = "1234567890"
  Debug.Print Len(miaStringa)
End Sub
```

Questo esempio visualizza, nella finestra Immediata, il valore `10`.

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStringa` e le assegna il valore `"1234567890"`. La procedura quindi visualizza il numero dei caratteri contenuti nella variabile `miaStr` servendosi della funzione `Len( )` inserita in un enunciato `Print`.

La chiamata della funzione `Len( )` prevede la presenza dell'argomento `miaStr` che specifica la stringa sulla quale operare.

LCase ( )

La funzione `LCase( )` converte i caratteri maiuscoli di una stringa in caratteri minuscoli.

Vedi anche: in questa parte, la sezione che descrive la funzione `UCase( )`.

La sintassi generale della funzione `LCase( )` è la seguente.

```
LCase(stringa)
```

La funzione `LCase( )` converte i caratteri maiuscoli dell'argomento stringa in caratteri minuscoli e restituisce la stringa risultante modificata. La funzione non influisce sui caratteri della stringa che non sono maiuscoli.

Per cambiare lo stato di maiuscolo/minuscolo dei caratteri di una variabile, si utilizzi tale variabile come valore per l'argomento `stringa` e si assegni il risultato ottenuto alla stessa variabile. Si ha anche la possibilità di modificare lo stato dei caratteri di una stringa servendosi sia della funzione `LCase( )` sia della funzione `UCase( )`.

Quello che segue è un esempio dell'uso della funzione LCase( ).

```
Private Sub Form_Load( )
  Dim miaStringa As String
  miaStringa = "SALVE, come va?"
  miaStringa = LCase(miaStringa)
  Debug.Print miaStringa
End Sub
```

Il risultato finale di questo esempio sarà la visualizzazione, nella finestra Immediata, di

```
salve, come va?
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStringa` e le assegna la sequenza di caratteri maiuscoli e minuscoli `"SALVE, come va?"`. La procedura chiama quindi la funzione `LCase( )` e le assegna come argomento la variabile `miaStringa`. La funzione chiamata converte la componente della stringa in caratteri maiuscoli in caratteri minuscoli generando, quindi, il risultato `"salve, come va?"`. A questo punto, il risultato ottenuto viene assegnato di nuovo alla variabile `miaStringa` il cui contenuto viene visualizzato nella finestra Immediata tramite l'enunciato `Print`.

LTrim ( ), RTrim ( ) e Trim ( )

Le funzioni `LTrim( )`, `RTrim( )` e `Trim( )` consentono di eliminare da una stringa la presenza di spazi eccessivi.

La sintassi generale per le funzioni `LTrim( )`, `RTrim( )` e `Trim` è la seguente.

```
LTrim(stringa)
RTrim(stringa)
Trim(stringa)
```

Il parametro stringa corrisponde alla stringa sulla quale la funzione dovrà operare. La funzione `LTrim( )` rimuove gli spazi in eccesso alla sinistra del parametro stringa e restituisce la stringa risultante. La funzione `RTrim( )` rimuove gli spazi in eccesso alla destra del parametro stringa e restituisce la stringa risultante. La funzione `Trim( )` è una combinazione delle due funzioni precedenti in quanto rimuove contemporaneamente dal parametro `stringa` gli spazi in eccesso a destra e a sinistra della stringa stessa restituendo la stringa elaborata finale.

Quello che segue è un esempio dell'uso combinato delle funzioni `LTrim( )`, `RTrim( )` e `Trim( )`.

```
Private Sub Form_Load( )
  Dim miaStringa As String
  miaStringa = "  Visual Basic  "
  Debug.Print "'"; LTrim(miaStringa); "'"
  Debug.Print "'"; RTrim(miaStringa); "'"
  Debug.Print "'";Trim(miaStringa); "'"
End Sub
```

Questo esempio visualizza, nella finestra Immediata, le seguenti linee:

```
'Visual Basic  '
'  Visual Basic'
'Visual Basic'
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStringa` e le assegna la stringa letterale `"  Visual Basic  "`. La procedura prevede tre enunciati `Print`, uno per ciascuna funzione di eliminazione degli spazi in eccesso.

Mid( )

La funzione `Mid( )` permette di estrarre alcuni caratteri da una stringa.

Vedi anche: in questa parte, le sezioni che descrivono le funzioni `Left( )` e `Right( )`.

La sintassi generale per la funzione `Mid( )` è la seguente.

```
Mid(stringa, inizio[, lunghezza])
```

Il parametro `stringa` corrisponde a un'espressione di tipo `String` (che può essere una costante stringa, una variabile o una loro combinazione) che contiene i caratteri da estrarre. Se il valore di questa espressione stringa è pari a `Null`, la funzione estrae `Null` (che corrisponde alla parola chiave utilizzata come sottotipo di `Variant` per indicare che una variabile `Variant` non contiene dati validi). Il parametro `inizio` corrisponde alla posizione del carattere nella stringa dal quale iniziare l'estrazione, se il parametro `inizio` eccede il numero dei caratteri della stringa, la funzione `Mid( )` restituisce una stringa vuota (`""`). Il parametro facoltativo `lunghezza`, invece, specifica il numero di caratteri da estrarre. Quando si omette l'indicazione del parametro lunghezza, o quando questo è inferiore al numero dei caratteri della stringa (compreso il carattere indicato da `inizio`) la funzione estrae tutti i caratteri a partire dalla posizione iniziale fino a quella finale della stringa.

Per maggiori informazioni sulla funzione `Mid( )`, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Quello seguente è un esempio di come usare la funzione `Mid( )`.

```
Private Sub Form_Load( )
  Dim miaStr As String
  miaStr = "1234567890"
  miaStr = Mid(miaStr, 3, 4)
  Debug.Print miaStr
End Sub
```

Questo esempio visualizza, nella finestra **Immediata**, il seguente risultato:

```
3456
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStr` e le assegna la stringa letterale `"1234567890"`. La procedura quindi estrae dalla variabile `miaStr` quattro caratteri a partire dal terzo servendosi, per questa operazione, della funzione `Mid( )`. Il risultato di questa estrazione viene memorizzato di nuovo all'interno della stessa variabile. La chiamata della funzione `Mid( )` prevede gli argomenti `miaStr, 3` e `4` che specificano, rispettivamente, la stringa dalla quale eseguire l'estrazione, la posizione iniziale per procedere con l'estrazione e il numero di caratteri da estrarre.

Replace ( )

La funzione `Replace( )` permette di sostituire tutte le occorrenze di una stringa all'interno di una stringa.

Vedi anche: in questa parte, le sezioni che descrivono le funzioni `Left( )` e `Right( )`.

La sintassi generale per la funzione `Replace( )` è la seguente.

```
Replace(stringa, ricerca, rimpiazzo[, inizio[, numero[,
confronto]]])
```

Il parametro `stringa` corrisponde a un'espressione di tipo `String` (che può essere una costante stringa, una variabile o una loro combinazione) che contiene la stringa completa. Il parametro `ricerca` specifica i caratteri da cercare e sostituire all'interno di `stringa`. Il parametro `rimpiazzo` specifica i caratteri con cui sostituire quelli cercati. Il parametro facoltativo `inizio` corrisponde alla posizione del carattere nella stringa dal quale iniziare la sostituzione. Quando esso viene omesso, la sostituzione comincia dal primo carattere

della stringa. Il parametro facoltativo `numero`, invece, specifica il numero massimo di occorrenze da sostituire. Quando esso viene omesso, la funzione sostituisce tutte le occorrenze trovate mentre l'argomento `confronto` specifica il tipo di confronto da avviare sulla stringa. L'argomento `confronto` può essere anche omesso, oppure potrà essere pari a -1, 0, 1 o 2. Il valore `0` (assunto per default) esegue un confronto di tipo binario; il valore `1` un confronto testuale (il valore -1 esegue un confronto utilizzando l'impostazione dell'istruzione Option Compare); il valore 2 (utilizzato solo per Microsoft Access) esegue un confronto in base alle informazioni contenute nel database.

Quello seguente è un esempio di come usare la funzione `Replace( )`.

```
Private Sub Form_Load( )
  Dim miaStr As String
  miaStr = "11011 Elm Street"
  Debug.Print miaStr
  miaStr = Replace(miaStr, "11", "22")
  Debug.Print miaStr
End Sub
```

Questo esempio visualizza, nella finestra Immediata, il seguente risultato:

```
11011 Elm Street
22022 Elm Street
```

La procedura `Form_Load( )` dichiara la variabile locale di tipo `String` `miaStr` e le assegna la stringa letterale `"11011 Elm Street"`, visualizzandolo nella finestra Immediata. La procedura quindi richiama la funzione `Replace( )` fornendole per parametri la stringa `miaStr` e le stringhe `"11"` e `"22"`. La funzione `Replace()` sostituisce tutte le occorrenze di `"11"` con `"22"` restituendo la variabile `miaStr` così ottenuta. Il contenuto della variabile `miaStr`, stampato nella finestra Immediata, è ora `"22022 Elm Street"`.

Right ( )

La funzione `Right( )` permette di estrarre alcuni caratteri da una stringa.

Vedi anche: in questa parte, le sezioni che descrivono le funzioni `Left( )` e `Mid( )`.

La sintassi generale per la funzione `Right( )` è la seguente.

```
Right(stringa, lunghezza)
```

Il parametro `stringa` corrisponde a un'espressione di tipo `String` che contiene i caratteri da estrarre. Se il valore di questa espressione stringa è pari a `Null`, la funzione estrae `Null` (che corrisponde alla parola chiave utilizzata come sottotipo di `Variant` per indicare che una variabile `Variant` non contiene dati validi). Il parametro `lunghezza`, invece, specifica il numero di caratteri da estrarre. Se l'argomento di questo parametro è pari a 0, la funzione restituisce una stringa vuota (`""`) e se l'argomento di questo parametro è pari o superiore al numero dei caratteri contenuti nella stringa, la funzione estrarrà tutti i caratteri.

Quello seguente è un esempio di come usare la funzione `Right( )`.

```
Private Sub Form_Load( )
  Dim miaStr As String
  miaStr = "1234567890"
  miaStr = Right(miaStr, 3)
  Debug.Print miaStr
End Sub
```

Questo esempio visualizza, nella finestra Immediata, il seguente risultato:

```
890
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `mia Str` e le assegna la stringa letterale `"1234567890"`. La procedura quindi estrae dalla variabile `miaStr` gli ultimi tre caratteri servendosi, per questa operazione, della funzione `Right( )`. Il risultato di questa estrazione viene memorizzato di nuovo all'interno della stessa variabile. La chiamata della funzione `Right( )` prevede gli argomenti `miaStr` e `3` che specificano, rispettivamente, la stringa dalla quale eseguire l'estrazione e il carattere dal quale iniziarla.

Nota. La funzione `Right(stringa,lunghezza)` si comporta nello stesso modo della funzione `Mid(stringa, Len(stringa) - lunghezza)`.

RTrim ( )

Vedi: in questa parte, la sezione "`LTrim( )`, `RTrim( )` e `Trim( )`".

Stringhe e loro manipolazione

Visual Basic prevede la presenza del tipo di dati `String` che consente di memorizzare, richiamare e manipolare i caratteri testuali. Si pensi a una stringa come a una matrice speciale di caratteri ai

quali si può singolarmente accedere tramite l'indicazione di un indice.

L'indice del primo carattere di una stringa corrisponde a "1" e le stringhe consentono ai programmi Visual Basic di memorizzare informazioni testuali come nomi, indirizzi, nomi di città e così via.

Vedi anche: in questa parte, la sezione che descrive come convertire le stringhe e altri tipi di dati.

La tabella seguente descrive i vari modi in cui è possibile manipolare le stringhe tenendo presente che tutti i comandi riportati nella tabella vengono anche singolarmente descritti in questa stessa parte.

Comando Visual Basic	*Descrizione*
`+`	Concatena fra loro due stringhe.
`InStr( )`	Ricerca la prima occorrenza di una stringa all'interno di un'altra stringa.
`InstrRev( )`	Ricerca l'ultima occorrenza di una stringa all'interno di un'altra stringa.
`Left( ), Right( ) e Mid( )`	Estrae i caratteri da una stringa partendo dall'inizio della stessa, da una posizione interna oppure dalla fine.
`Len( )`	Restituisce la lunghezza di una stringa.
`LCase( ), UCase( )`	Converte i caratteri di una stringa, rispettivamente, in caratteri minuscoli o maiuscoli.
`LTrim( ), RTrim( ) e Trim( )`	Elimina gli spazi aggiuntivi, rispettivamente, a sinistra a destra o da entrambi i lati della stringa.
`Replace( )`	Sostituisce le occorrenze di una stringa all'interno di un'altra stringa.
`StrReverse( )`	Inverte i caratteri presenti in una stringa.

StrReverse ( )

La funzione `StrReverse( )` sposta di posizione tutti i caratteri all'interno di una stringa in modo tale che il primo diventi l'ultimo e, viceversa, l'ultimo diventi il primo.

La sintassi generale della funzione `StrReverse( )` è la seguente.

StrReverse(stringa)

La funzione `StrReverse( )` restituisce in ordine inverso i caratteri presenti nel parametro `stringa`.

Quello che segue è un esempio dell'uso della funzione `StrReverse( )`.

```
Private Sub Form_Load( )
  Dim miaStringa As String
  miaStringa = "Salve, come va?"
  miaStringa = StrReverse(miaStringa)
  Debug.Print miaStringa
End Sub
```

Il risultato finale di questo esempio sarà la visualizzazione, nella finestra Immediate, di

```
?av emoc ,evlaS
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStringa` e le assegna la sequenza di caratteri `"Salve, come va?"`. La procedura chiama quindi la funzione `StrReverse( )` e le assegna come argomento la variabile `miaStringa`. La funzione chiamata inverte i caratteri della stringa generando, quindi, il risultato `"?av emoc ,evlaS "`. A questo punto, il risultato ottenuto viene assegnato di nuovo alla variabile `miaStringa` il cui contenuto viene visualizzato nella finestra Immediata tramite l'enunciato `Print`.

Trim( )

Vedi anche: in questa parte, la sezione "`LTrim( )`, `RTrim( )` e `Trim( )`".

UCase( )

La funzione `UCase( )` converte i caratteri minuscoli di una stringa in caratteri maiuscoli.

Vedi anche: in questa parte, la sezione che descrive la funzione `LCase( )`.

La sintassi generale della funzione `UCase( )` è la seguente.

UCase(stringa)

La funzione `UCase( )` converte i caratteri minuscoli dell'argomento `stringa` in caratteri maiuscoli e restituisce la stringa risultante

modificata. La funzione non influisce sui caratteri della stringa che non sono minuscoli.

Quello che segue è un esempio dell'uso della funzione `UCase( )`.

```
Private Sub Form_Load( )
  Dim miaStringa As String
  miaStringa = "SALVE, come va?"
  miaStringa = UCase(miaStringa)
  Debug.Print miaStringa
End Sub
```

Il risultato finale di questo esempio sarà la visualizzazione, nella finestra Immediata, di

```
SALVE, COME VA?
```

La procedura `Form_Load( )` dichiara la variabile stringa locale `miaStringa` e le assegna la sequenza di caratteri maiuscoli e minuscoli `"SALVE, come va?"`. La procedura chiama quindi la funzione `UCase( )` e le assegna come argomento la variabile `miaStringa`. La funzione chiamata converte la componente della stringa in caratteri minuscoli in caratteri maiuscoli generando, quindi, il risultato `"SALVE, COME VA?"`. A questo punto, il risultato ottenuto viene assegnato di nuovo alla variabile `miaStringa` il cui contenuto viene visualizzato nella finestra Immediata tramite l'enunciato `Print`.

Gestione dei form e form MDI

I form MDI (*Multiple Document Interface*) permettono di migliorare sensibilmente l'aspetto dei propri programmi, consentendo di mantenere aperti contemporaneamente, nell'applicazione, più documenti. La struttura prevede un form principale dal quale dipendono uno o più form secondari.

Argomenti trattati

- **I form MDI (*Multiple Document Interface*)**
- **Gestione di form multipli**
- **Form principali e dipendenti (Form MDI)**

Aggiunta di form MDI secondari

Un form MDI principale può prevedere la presenza di uno o più form figli così come vengono definiti nella struttura della finestra Object Explorer. Questa finestra contiene sempre almeno un form figlio il che significa che un form figlio prevede almeno un form MDI principale dal quale parte l'esecuzione dell'applicazione. In un progetto si potranno creare altri form figli sui quali avviare la varie operazioni. In altre parole, se si desidera creare un'applicazione composta da un massimo di quattro form figli, il progetto dovrà prevedere lo stesso numero di form figli (e ognuno di questi avrà un proprio nome tramite il quale si potrà accedere al form stesso).

Questo scenario di un form principale e un certo numero fisso di form secondari è, indubbiamente, il più facile ma, di contro, limita le possibilità dell'applicazione MDI nel suo complesso. Uno scenario più flessibile si può ottenere con una condizione che preveda un numero variabile di form secondari e dove l'utente dell'applicazione possa decidere quanti form MDI secondari utilizzare.

Per gestire form secondari di un progetto, è necessario utilizzare una delle seguenti variabili a livello di form.

- Un oggetto di tipo `Collection` che memorizzi i form secondari come oggetti.
- Un indice di form secondari che mantenga una traccia del numero totale dei form secondari che verranno via via creati. Il programma incrementa il valore di questo indice ogni volta che verrà creato un nuovo form secondario. Quando il programma chiuderà il form figlio precedentemente aperto, il valore dell'indice non subirà alcuna variazione. Questo schema permette all'indice di fornire sempre un identificativo univoco per un form secondario. Di solito, tale numero identificativo viene utilizzato come parte dell'etichetta descrittiva del form stesso.

Per aggiungere a un progetto un form secondario è necessario effettuare le seguenti operazioni.

- Dichiarare una variabile locale che crei un nuovo form secondario.
- Incrementare di `1` il valore dell'indice del form secondario.
- Caricare il form secondario servendosi dell'enunciato `Load object`.
- Intervenire sulle proprietà del nuovo form secondario (come, per esempio, impostare la proprietà `Caption`).

- Visualizzare il nuovo form secondario tramite il metodo `Show( )`.
- Aggiungere il nuovo form secondario alla serie di oggetti servendosi del metodo `Add(oggettoDaAggiungere)`.

Viene ora proposto un esempio di un gestore di evento (associato al comando `New`) che crea e visualizza un nuovo form secondario.

```
Private Sub mnuNew_Click( )
  Dim MDIchildForm as New MDIchild
  ' incrementa la variabile di livello del form
  MDIchildIndex = MDIchildIndex + 1
  Load MDIchildForm
   MDIchildForm.Caption = "MDI dipendente" +
Str(MDIchildIndex)
  MDIchildForm.Show
  colMDIforms.Add MDIchildForm
End Sub
```

Questo gestore di evento esegue le operazioni seguenti.

- Dichiara una variabile locale `MDIchildForm` che crea un nuovo form secondario (del tipo `MDIchild`).
- Incrementa di 1 l'indice `MDIchildIndex` del form secondario.
- Carica il form servendosi dell'enunciato `Load MDIchildForm`.
- Interviene sulla proprietà `Caption` del form in modo da inserirvi il valore dell'indice `MDIchildIndex`.
- Visualizza, servendosi del metodo `Show( )`, il nuovo form.
- Aggiunge il nuovo form alla serie di oggetti `colMDIforms`, servendosi, per questa operazione, del metodo `Add( )`. Il valore dell'argomento per questo metodo corrisponde a `MDIchildForm`.

Sistemazione di form MDI dipendenti

Visual Basic permette di gestire form o icone all'interno di un oggetto `MDIForm` utilizzando il metodo `Arrange( )`.

La sintassi generale per l'adozione di questo metodo è la seguente.

```
MDIparentForm.Arrange tipoSistemazione
```

`MDIparentForm` corrisponde al nome del form MDI principale e l'argomento `tipoSistemazione` corrisponde a un valore o costan-

te che specifica come sistemare il form o le icone all'interno di un oggetto form MDI principale. I valori per l'argomento `tipoSistemazione` sono i seguenti.

- La costante `vbCascade` (alla quale viene assegnato il valore `0`) sovrappone l'uno all'altro tutti i form MDI dipendenti non ridotti a icona.
- La costante `vbTileHorizontal` (alla quale viene assegnato il valore `1`) affianca orizzontalmente tutti i form MDI dipendenti non ridotti a icona.
- La costante `vbTileVertical` (alla quale viene assegnato il valore `2`) affianca verticalmente tutti i form MDI dipendenti non ridotti a icona.
- La costante `vbArrangeIcons` (alla quale viene assegnato il valore `3`) sistema le icone dei form MDI dipendenti ridotti a tale stato.

Vengono ora proposti vari esempi di gestori di evento destinati a organizzare la disposizione di form MDI dipendenti.

```
Private Sub mnuArrangeIcons_Click( )
  MDIparent.Arrange vbArrangeIcons
End Sub
Private Sub mnuCascade_Click( )
  MDIparent.Arrange vdCascade
End Sub
Private Sub mnuHTile_Click( )
  MDIparent.Arrange vbTileHorizontal
End Sub
Private Sub mnuVTile_Click( )
  MDIparent.Arrange vbTilevertical
End Sub
```

La procedura `mnuArrangeIcons_Click( )` sistema le icone dei form MDI dipendenti applicando il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento `tipoSistemazione` è, in questo caso, `vbArrangeIcons`.

La procedura `mnuCascade_Click( )` sovrappone tutti i form MDI dipendenti non ridotti a icona applicando il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento `tipoSistemazione` è, in questo caso, la costante `vbCascade`.

La procedura `mnuVTile_Click( )` affianca verticalmente tutti i form secondari non ridotti a icona applicando il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento `tipoSistemazione` è, in questo caso, la costante `vbTileVertical`.

La procedura `mnuHTile_Click( )` affianca orizzontalmente tutti i form MDI dipendenti non ridotti a icona applicando il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento `tipoSistemazione` è, in questo caso, la costante `vbTileHorizontal`.

Chiusura del form secondario attivo

Per chiudere il form secondario attivo è necessario accedere alla proprietà `ActiveForm` del form MDI principale. Quando si usa un oggetto `Collection` per la gestione dei form MDI dipendenti, sarà necessario utilizzare una procedura che esegua le operazioni seguenti.

- Ricerca del form attivo tramite un ciclo che confronti, per esempio, la proprietà `Caption` di una voce nell'oggetto `Collection` con la proprietà `parentMDI.ActiveForm.Caption`. Quando i valori delle due proprietà `Caption` coincidono, la procedura dovrebbe procedere con l'esecuzione delle operazioni successive.
  - Scarica il form MDI secondario corrispondente servendosi dell'enunciato `MDIchildForm`.
  - Rimuove il form MDI secondario attivo dall'oggetto `Collection` servendosi, per questa operazione, del metodo `Remove(indice)`.

Si propone ora un esempio che mostra un gestore di evento che risponde al clic sul menu `Chiudi` associato al menu della finestra MDI principale `MDIparent`.

```
Private Sub mnuClose_Click( )
  Dim I As Integer
  ' Ricerca il form MDI attivo
  For I = 1 To colMDIforms.Count
    ' confronta le proprietà Caption
    If colMDIforms.Item(I).Caption =
MDIparent.AcriveForm.Caption Then
      ' trovato il form MDI attivo!!
      ' scarica il form MDI attivo
      Unload colMDIforms.Item(I)
      ' rimuove il form MDI dalla serie
      colMDIforms.Remove I
      ' esce dopo aver rimosso il form
      ' MDI secondario
      Exit Sub
    End If
  Next I
End Sub
```

Questo gestore di evento dichiara la variabile `I` da utilizzare come variabile di un ciclo. La procedura utilizza un ciclo `For` per reperire un form attivo all'interno dell'oggetto `Collection` utilizzato per memorizzare i form MDI secondari.

Il ciclo contiene un enunciato `If` che confronta il contenuto della proprietà `Caption` di ogni form MDI secondario (ai quali accede utilizzando l'espressione `colMDIforms.ActiveForm.Caption`) e della proprietà `Caption` del form MDI secondario attivo (al quale accede tramite l'espressione `MDIparent.ActiveForm.Caption`). Quando le due proprietà coincidono, l'enunciato `If` esegue le seguenti operazioni.

- Scarica il form MDI secondario attivo servendosi, per questa operazione, dell'enunciato `Unload.colMDIforms.Item(I)`.
- Rimuove dall'oggetto `Collection` il form MDI dipendente attivo servendosi del metodo `Remove( )`. Il valore dell'argomento di questo metodo corrisponde alla variabile `I` (ovvero all'indice del form MDI secondario attivo).
- Esce dalla procedura utilizzando l'enunciato `Exit Sub`.

Chiusura di tutti i form secondari

Chiudere tutti i form MDI secondari è un'operazione particolarmente semplice se si utilizza, per la gestione raggruppata di tutti i form secondari, l'oggetto `Collection`. È necessario, tuttavia, avere a disposizione una routine che contenga un ciclo `For` a conteggio a ritroso il cui scopo è quello di elaborare le voci dell'oggetto `Collection`, eseguendo le seguenti operazioni.

- Scaricare i form secondari tramite l'enunciato `Unload MDIchildForm`.
- Rimuovere i form secondari dall'oggetto `Collection` servendosi, per questa operazione, del metodo `Remove(indice)`.

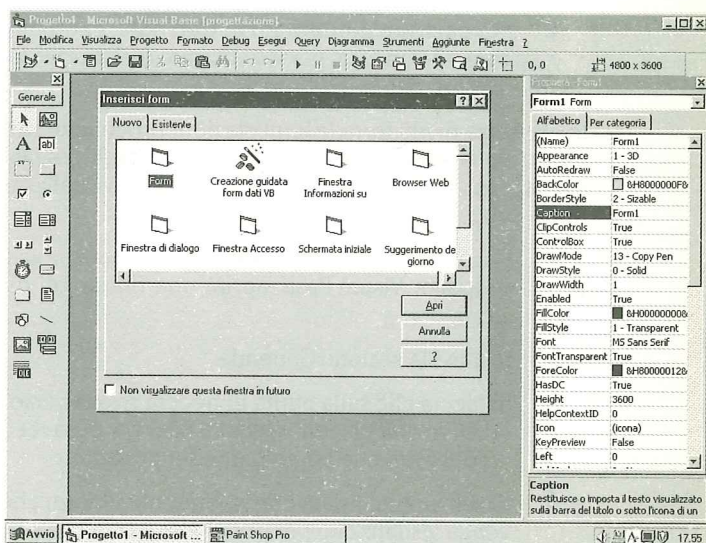
L'esempio che viene ora proposto mostra un gestore di evento che risponde al clic sul comando di menu `Chiudi Tutti` associato al menu del form MDI principale `MDIparent`.

```
Private Sub mnuCloseAll_Click( )
  Dim I As Integer
  For I = colMDIforms.Count To 1 Step -1
    ' scarica il form MDI
    Unload colMDIforms.Item(I)
    ' rimuove il form MDI dalla serie dei form
    volMDIforms.Remove I
  Next I
End Sub
```

Questo gestore di evento dichiara la variabile `I` come variabile di controllo di un ciclo. La procedura utilizza un ciclo `For` a conteggio regressivo per avviare un'iterazione sui form MDI dipendenti dell'oggetto `Collection` (ovvero agisce dall'ultimo al primo). Gli enunciati del ciclo eseguono le seguenti operazioni.

- Scaricano un form secondario servendosi dell'enunciato `Unload.colMDIforms.Item(I)`.
- Rimuovono un form secondario dall'oggetto `Collection colMDIforms` servendosi, per questa operazione, del metodo `Remove( )`. Il valore dell'argomento di questo metodo corrisponde alla variabile `I`.

Gestione di più form

Visual Basic permette di creare un progetto composto da più form che potranno essere generici oppure di tipo specialistico. Quando si richiama il comando Inserisci form del menu Progetto, Visual Basic visualizza la finestra corrispondente, simile a quella della figura seguente.

Visual Basic è in grado di riconoscere e gestire i seguenti form di tipo specialistico.

- **Finestra di dialogo "Informazioni su"** che visualizza le informazioni relative all'applicazione.
- **Finestra di dialogo "Accesso"** che richiede all'utente di digitare il proprio identificativo e la propria parola d'ordine prima di procedere con l'esecuzione dell'applicazione.
- **Finestra di dialogo "Opzioni"**, suddivisa in sezioni contenenti le varie opzioni per la personalizzazione dell'applicazione.
- **Videata "Schermata iniziale"** nella quale si potrà inserire il nome dell'applicazione, quello del programmatore, un logo e qualsiasi altro elemento informativo che si desidera trasmettere all'utente dell'applicazione stessa (si tratta, in sostanza, di una videata di tipo "commerciale").
- **Videata "Suggerimento del giorno"** che visualizza un consiglio operativo ogni volta che l'utente avvia l'applicazione.
- **Videata "ODBC Login"** che presenta le opzioni per connettersi a un database.
- **"Browser Web"** per aggiungere all'applicazione una pagina Web Internet.
- **"Creazione guidata form dati VB"** che genera automaticamente un form per visualizzare o elaborare le informazioni di un database.

In presenza di più form, le impostazioni predefinite caricheranno il primo della serie (al quale viene per default assegnato il nome `Form1`); per selezionare un form diverso da visualizzare come primo, si proceda come segue.

1. Si acceda al comando Proprietà di Nome progetto del menu Progetto; Visual Basic aprirà una finestra di dialogo di opzioni.
2. Si attivi la sezione Generale.
3. Si faccia clic sulla punta di freccia rivolta verso il basso dell'elenco dell'opzione Oggetto di avvio per accedere all'elenco dei form che fanno parte del progetto.
4. Si selezioni il nome corrispondente al form che si desidera venga visualizzato per primo e si faccia clic su OK per confermare.

Per maggiori informazioni su come lavorare con molti form, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Form MDI (Multiple Document Interface)

Un'applicazione MDI (*Multiple Document Interface*) è un tipo di programma che permette di lavorare contemporaneamente con più di un documento. I programmi di elaborazione testo come Microsoft Word per Windows e WordPerfect per Windows sono degli ottimi esempi di applicazioni di tipo MDI.

Visual Basic consente la creazione di applicazioni MDI per le quali, tuttavia, è necessario avere un solo form MDI principale e almeno un form secondario. Visual Basic utilizza il termine "MDIForm" per indicare un form MDI *principale* e per creare o inserire un "MDIForm" si dovrà richiamare il comando Inserisci form MDI dal menu Progetto.

Poiché in Visual Basic si possono creare form secondari, in un unico progetto si potranno avere uno o più form la cui proprietà `MDIchild` sia impostata su `True`. In sostanza, è questa l'unica operazione necessaria per creare form secondari.

Se un form secondario prevede la presenza di propri menu, ogni volta che uno di questi form risulterà attivo, la corrispondente Barra dei menu andrà automaticamente a sostituire quella dell'oggetto `MDIForm`. L'icona di un form secondario verrà visualizzata all'interno del form MDI principale tenendo presente che un form `MDIForm` può contenere solo controlli di tipo `Menu` e `PictureBox` e altri controlli di tipo personalizzato che prevedano una proprietà `Align`. Per inserire in un oggetto `MDIForms` altri tipi di controllo sarà necessario creare una struttura per un'immagine e quindi inserire nella stessa i controlli desiderati. Per visualizzare il testo di una struttura di immagine di un `MDIForm` si potrà usare il metodo `Print` ma lo stesso metodo non potrà essere usato per visualizzare il testo contenuto nell'oggetto `MDIForm` stesso. La progettazione dei form secondari viene eseguita indipendentemente dall'oggetto `MDIForm` ma, in fase di esecuzione dell'applicazione, i form dipendenti creati risulteranno sempre contenuti all'interno del form principale.

Per maggiori informazioni su come lavorare con form MDI, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998.).

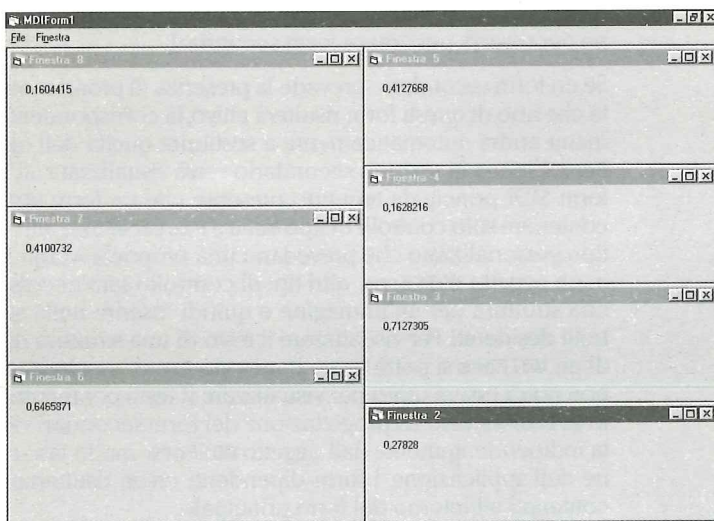
Gestione dei form MDI

La gestione dei form principali prevede l'esecuzione delle seguenti operazioni.

- Uso delle variabili a livello form per gestire le informazioni correlate ai form secondari.

- ✦ Uso dei gestori di evento per rispondere al sistema di menu che è parte integrante del form MDI principale.
- ✦ Uso di eventi speciali, come `Load( )` e `Unload( )` per la gestione del caricamento e scaricamento del form MDI principale. Questi eventi consentono di inizializzare i form secondari e di scaricarli quando si chiude l'applicazione con form MDI.

Si ha anche la possibilità di gestire altri eventi associati al form MDI principale quali, per esempio, quelli relativi alle operazioni eseguite con il mouse.

L'esempio che segue, il cui risultato finale è rappresentato dalla figura, utilizza un form MDI principale per la gestione dei form MDI dipendenti.

La tabella seguente mostra il sistema dei menu del form MDI principale (quello al quale è stato assegnato il nome `MDIparent`). Inizialmente il form visualizza solo un form secondario e ogni form secondario visualizzerà, ogni secondo, un numero casuale.

Menu	Comando	Name	Caption
File		mnuFile	&File
	Nuovo	mnuNew	&Nuovo
	Chiudi tutti	mnuCloseAll	Chiudi &Tutti
	Chiudi	mnuClose	&Chiudi
	(separatore)	N1	-
	Esci	mnuExit	&Esci
Window		mnuWindow	Finestra
	Affianca orizzontalmente	mnuVTile	Affianca &Orizzontalmente
	Affianca verticalmente	mnuHTile	Affianca &Verticalmente
	Sovrapponi	mnuCascade	&Sovrapponi
	Disponi icone	mnuArrangeIcons	&Disponi

L'utente potrà utilizzare il comando `Nuovo` del menu `File` per aggiungere altri form secondari e il comando `Chiudi`, sempre del menu `File`, per chiudere il form secondario attivo. Il comando `Chiudi Tutti` del menu `File` chiuderà tutti i form secondari. I comandi del menu `Finestra` hanno lo scopo, invece, di determinare la disposizione a video dei form secondari. Nella tabella seguente si riportano le impostazioni delle proprietà dei vari oggetti che fanno parte del progetto.

Oggetto	Proprietà	Impostazione
Form	Name	MDchild
Etichetta	Name	lblNumber
Temporizzatore	Name Interval	tmtTimer 1000

Ecco il codice sorgente che genera l'applicazione.

```
Option Explicit
Dim MDIchildIndex As Integer
Dim colMDIforms As New Collection

Private Sub MDIForm_Load( )
  MDIchild.Caption = "Finestra 1"
  colMDIforms.Add MDIchild
  ' imposta il contatore dei form secondari
  MDIchildIndex = 1
End Sub
Private Sub mnuArrangeIcons_Click( )
  MDIparent.Arrange vbArrangeIcons
End Sub
```

```
Private Sub mnuCascade_Click( )
  MDIparent.Arrange vbCascade
End Sub
Private Sub mnuClose_Click( )
  Dim I As Integer
  ' ricerca il form MDI attivo
  For I = 1 To colMDIforms.Count
    ' confronta le proprietà Caption
    If colMDIforms.Item(I).Caption =
MDIparent.ActiveForm.Caption Then
      ' trovato il form MDI attivo!!
      ' scarica il form attivo
      Unload colMDIforms.Item(I)
      ' rimuove il form MDI dalla serie
      colMDIforms.Remove I
      ' esce dopo aver eliminato il form
      ' secondario
      Exit Sub
    End If
  Next I
End Sub
Private Sub mnuCloseAll_Click( )
  Dim I As Integer
  For I = colMDIforms.Count To 1 Step -1
    'scarica il form MDI
    Unload colMDIforms.Item(I)
    ' rimuove il form MDI dalla serie
    colMDIforms.Remove I
  Next I
End Sub
Private Sub mnuExit_Click( )
  End
End Sub
Private Sub mnuHTile_Click( )
  MDIparent.Arrange vbTileHorizontal
End Sub
Private Sub mnuNew_Click( )
  Dim MDIchildForm As New MDIchild
  MDIchildIndex = MDIchildIndex + 1
  Load MDIchildForm
  MDIchildForm.Caption = "Finestra" + Str(MDIchildIndex)
  MDIchildForm.Show
  colMDIforms.Add MDIchildForm
End Sub
Private Sub mnuVTile_Click( )
  MDIparent.Arrange vbTileVertical
End Sub
```

L'esempio dichiara le seguenti variabili a livello form.

- La variabile MDIchildIndex, di tipo Integer che corrisponde all'indice del form secondario.
- L'oggetto colMDIforms, di tipo Collection, che memorizza i form secondari.

La procedura MDIForm_Load( ) gestisce il caricamento del form MDI principale, eseguendo le seguenti operazioni.

- Assegna, alla proprietà Caption del primo form secondario, la stringa letterale "Finestra 1".
- Aggiunge il primo form secondario all'oggetto colMDIforms. Questa operazione si serve del metodo Add e il valore dell'argomento di tale metodo corrisponde al nome del form secondario MDIchild.
- Assegna il valore 1 all'indice MDIchildIndex.

Il gestore di evento mnuNew_Click( ) risponde al clic sul comando Nuovo eseguendo le operazioni seguenti.

- Dichiara una variabile locale MDIchildForm che crea un nuovo form secondario (di tipo MDIchild).
- Aggiunge 1 all'indice MDIchildIndex del form secondario.
- Carica il form secondario servendosi dell'enunciato Load MDIchildForm.
- Imposta la proprietà Caption del nuovo form secondario in modo che comprenda il valore dell'indice MDIchildIndex.
- Visualizza il nuovo form secondario servendosi del metodo Show( ).
- Aggiunge il nuovo form secondario all'oggetto colMDIforms servendosi, per questa operazione, del metodo Add( ). Il valore dell'argomento per questo metodo corrisponde a MDIchildForm.

La procedura mnuClose_Click( ) risponde al clic sul comando Chiudi. Questo gestore di evento dichiara la variabile I che dovrà essere usata come variabile di controllo per un ciclo. La procedura utilizza un ciclo For per reperire il form attivo nell'oggetto Collection utilizzato per memorizzare i form secondari. Il ciclo contiene un enunciato If che confronta il contenuto della proprietà Caption di ogni form secondario (alla quale si accede servendosi dell'espressione colMDIforms.Item(I).Caption) e quello della proprietà Caption del form secondario attivo (alla quale si accede

tramite l'espressione `MDIparent.ActiveForm.Caption`). Quando i valori delle due proprietà coincidono, l'enunciato `If` eseguirà le seguenti operazioni.

- Scarica il form MDI dipendente attivo servendosi dell'enunciato `Unload colMDIforms.Item(I)`.
- Rimuove, dall'oggetto `Collection colMDIforms` il form secondario servendosi, per questa operazione, del metodo `Remove( )`. Il valore dell'argomento per questo metodo corrisponde alla variabile `I` (ovvero all'indice del form secondario attivo).
- Esce dalla procedura utilizzando l'enunciato `Exit Sub`.

La procedura `mnuCloseAll_Click( )` risponde al clic sul comando `Chiudi Tutto` dichiarando la variabile `I` da utilizzare come variabile di controllo del ciclo. La procedura si serve di un ciclo `For` a conteggio a ritroso per avviare un'iterazione sui form secondari contenuti nell'oggetto `Collection` (procedendo dall'ultimo al primo). Gli enunciati del ciclo eseguono le seguenti operazioni.

- Scaricamento di un form secondario servendosi dell'enunciato `colMDIforms.Item(I)`.
- Rimozione del form secondario dall'oggetto `Collection colMDIforms` servendosi, per questa operazione, del metodo `Remove( )`. Il valore dell'argomento per tale metodo corrisponde alla variabile `I`.

La procedura `mnuArrangeIcons_Click( )` risponde al clic sul comando `Disponi Icone` che sistema le icone dei form secondari ridotti a icona applicando il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento relativo alla sistemazione delle icone corrisponde alla costante `vbArrangeIcons`.

La procedura `mnuCascade_Clic( )` risponde al clic sul comando `Sovrapponi` che sovrappone tutte le finestre dei form secondari non ridotte a icona applicando, per questa operazione, il metodo `Arrange( )` al form MDI principale `MDIparent`. Il valore dell'argomento per la sistemazione degli oggetti corrisponde alla costante `vbCascade`.

La procedura `mnuVTile_Click( )` risponde al clic sul comando `Affianca Verticalmente` che affianca verticalmente tutte le finestre dei form secondari non ridotte a icona servendosi, per questa operazione, del metodo `Arrange( )` applicato al form MDI principale `MDIparent`. Il valore per l'argomento per la sistemazione degli oggetti corrisponde alla costante `vbTileVertical`.

La procedura `mnuHTile_Click( )` risponde al clic sul comando `Affianca Orizzontalmente` che affianca orizzontalmente tutte le finestre dei form secondari non ridotte a icona servendosi, per questa operazione, del metodo `Arrange( )` applicato al form MDI principale `MDIparent`. Il valore per l'argomento per la sistemazione degli oggetti corrisponde alla costante `vbTileHorizontal`.

Manipolazione e accesso alle matrici

Le matrici (o *array*) sono dei contenitori vuoti nei quali è possibile memorizzare, almeno virtualmente, qualsiasi cosa, sempre che sia composta da tipi di dati accettabili. Le matrici possono essere monodimensionali (e in questo caso conterranno solo un lungo elenco di elementi), bidimensionali (come nel caso di una tabella composta da righe e colonne) o tridimensionali (assumendo, in questo caso, la forma di un ideale "cubo" di contenimento dei dati). In effetti, si ha la possibilità anche di creare matrici multidimensionali composte da un massimo di 60 dimensioni (anche se questa disposizione non è visivamente immaginabile).

Argomenti trattati

- ✔ **Uso delle matrici mono e multidimensionali per la memorizzazione dei dati**
- ✔ **Come estrarre i dati dalle matrici mono e multidimensionali**
- ✔ **Dichiarazione delle matrici**
- ✔ **Uso delle matrici come funzioni o procedure**

Limiti delle matrici

Visual Basic consente di accedere agli indici inferiore e superiore di una matrice monodimensionale tramite le funzioni predefinite LBound( ) e UBound( ). Queste funzioni consentono di iterare tutti gli elementi di una matrice.

La sintassi generale per le funzioni LBound( ) e Ubound( ) è la seguente.

```
LBound(nomeMatrice)
UBound(nomeMatrice)
```

La funzione LBound( ) restituisce il valore inferiore dell'indice della matrice mentre la funzione UBound( ) restituisce quello superiore.

La sezione di questa parte che descrive l'accesso alle matrici monodimensionali contiene un esempio pratico dell'uso delle funzioni LBound( ) e UBound( ).

Per accedere agli indici inferiore e superiore di una matrice multidimensionale si utilizza, invece, un'altra versione delle funzioni predefinite LBound( ) e UBound( ). Queste funzioni alternative consentono di avviare un'iterazione su tutti gli elementi di una matrice multidimensionale.

La sintassi generale per tali funzioni è la seguente.

```
LBound(nomeMatrice, numeroDimensione)
UBound(nomeMatrice, numeroDimensione)
```

La funzione LBound( ) utilizzata in una matrice multidimensionale restituisce il valore inferiore dell'indice della dimensione specificata dall'argomento numeroDimensione mentre la funzione UBound( ) restituisce quello superiore. Il numero minimo applicabile all'argomento numeroDimensione è 1.

Nella sezione di questa parte che descrive come accedere alle matrici multidimensionali vi è un esempio pratico dell'uso di queste due funzioni.

Accesso a matrici multidimensionali

Le matrici multidimensionali consentono di memorizzare valori e richiamarli da una struttura precedentemente definita.

La sintassi generale per accedere a un elemento di una matrice multidimensionale è la seguente.

```
nomeArray(indice1, indice2[, indice3, ...])
```

Questa sintassi indica che è necessario specificare innanzitutto il nome della matrice seguito da una serie di indici racchiusi tra parentesi tonde. Il numero dell'indice coincide con la dimensione della matrice; inoltre, i valori degli indici devono essere compresi nell'intervallo che è stato utilizzato per dichiarare la matrice stessa.

Per accedere all'indice inferiore o superiore di una matrice multidimensionale si utilizzano le funzioni predefinite `LBound( )` e `UBound( )` la cui sintassi generale è la seguente.

```
LBound(nomeMatrice, numeroDimensione)
UBound(nomeMatrice, numeroDimensione)
```

La funzione `LBound( )` restituisce il valore dell'indice inferiore della dimensione specificata dall'argomento `numeroDimensione` mentre la funzione `UBound( )` restituisce il valore superiore. Si tenga comunque presente che il valore di dimensione minimo ammissibile è pari a 1.

Si fornisce ora un esempio che consente di accedere ai dati di una matrice bidimensionale.

```
Private Sub Form_Load( )
  Dim Arr(10, 10) As Double
  Dim Num As Variant
  Dim Somma As Double
  Dim Riga As Integer
  Dim Col As Integer
  Randomize
  ' memorizza in una matrice una serie di valori casuali
  For Riga = LBound(Arr, 1) To UBound(Arr, 1)
    For (Col = LBound(Arr, 2) To UBound(Arr, 2)
      Arr(Riga, Col) = 1000 * Rnd( )
    Next Col
  Next Riga
  ' somma iniziale
  Somma = 0
  ' aggiunge alla somma gli elementi della matrice
  For Each Num In Arr
    Somma = Somma + Num
  Next Num
  ' Visualizza la somma
  Debug.Print "Somma = "; Somma
End Sub
```

Il codice potrebbe presentare, come risultato, la seguente linea:

```
Somma = 64918.2204604149
```

Questo esempio dichiara la matrice bidimensionale `Arr` di tipo `Double` alla quale vengono assegnate 11 righe e 11 colonne, la procedura `Form_Load( )` si serve di un ciclo `For-Next` per assegnare agli elementi della matrice `Arr` una serie di numeri casuali. Questi cicli fanno uso delle funzioni predefinite `LBound( )` e `UBound( )` per accedere ai limiti inferiore e superiore di ogni dimensione della matrice `Arr( )`. Per esempio, l'espressione `LBound(Arr, 1)` e `UBound(Arr, 1)` restituiranno, rispettivamente, i valori 0 e 10 che, in effetti, corrispondono agli indici inferiore e superiore della prima dimensione della matrice in esame.

La procedura `Form-Load( )` si serve di un semplice ciclo `For Each` per aggiungere i valori nella matrice `Arr( )` alla variabile `Somma` visualizzandone, successivamente, il valore nella finestra Immediata.

Accesso a matrici monodimensionali

L'accesso agli elementi di una matrice monodimensionale consente di memorizzare ed estrarre valori da un elemento matrice.

La sintassi generale per accedere a un elemento di una matrice monodimensionale è la seguente.

```
nomeMatrice(indice)
```

Questa sintassi mostra che è necessario prima specificare il nome della matrice seguito da un indice racchiuso tra parentesi tonde (il cui scopo è quello di selezionare un elemento della matrice stessa). Il valore per l'indice deve essere compreso nell'intervallo definito quando si è dichiarata la matrice.

Per accedere agli indici inferiore o superiore di una matrice monodimensionale, si devono utilizzare le funzioni predefinite `LBound( )` e `UBound( )` la cui sintassi generale è la seguente.

```
LBound(nomeMatrice)
UBound(nomeMatrice)
```

La funzione `LBound( )` restituisce il valore dell'indice inferiore della matrice `nomeMatrice` mentre la funzione `UBound( )` restituisce quello superiore.

L'esempio che segue indica come accedere a una matrice monodimensionale.

```
Private Sub Form_Load( )
  Dim Arr(10) As Double
  Dim Num As Variant
  Dim Somma As Double
```

```
  Dim I As Integer
  Randomize
  ' memorizza in una matrice una serie di valori casuali
  For I = LBound(Arr) To UBound(Arr)
    Arr(I) = 1000 * Rnd( )
  Next I
  ' somma iniziale
  Somma = 0
  ' aggiunge alla somma un elemento della matrice
  For Each Num In Arr
    Somma = Somma + Num
  Next Num
  ' visualizza il risultato della somma
  Debug.Print "Somma = "; Somma
End Sub
```

Questo codice potrebbe visualizzare, nella finestra Immediate la seguente linea:

```
Somma = 4880.45918941498
```

Il codice dichiara la matrice `Arr` di tipo `Double` contente 11 elementi. La procedura `Form_Load( )` utilizza un ciclo `For-Next` per assegnare agli elementi della matrice `Arr` una serie di numeri casuali. Il ciclo `For-Next` si serve delle funzioni predefinite `LBound( )` e `UBound( )` per accedere ai limiti inferiore e superiore della matrice `Arr( )`. Le espressioni `LBound(Arr)` e `UBound(Arr)` restituiscono, rispettivamente, i valori 0 e 10 che corrispondono agli indici inferiore e superiore della matrice stessa.

La procedura `Form_Load( )` utilizza un solo ciclo `For Each` per aggiungere alla variabile `Somma` i valori della matrice `Arr( )` visualizzandone successivamente il risultato nella finestra Immediata.

Copia di matrici

Visual Basic consente di copiare matrici usando l'operatore =, esattamente come per gli altri tipi di variabili. Occorre comunque attenersi a due semplici regole.

- Le matrici a dimensione variabile possono apparire da entrambe le parti dell'operatore di assegnazione =.
- Le matrici di dimensione fissa possono apparire solo alla destra dell'operatore di assegnazione =.

ESEMPIO EXAMPLE

Ecco un esempio di codice per la copia di matrici.

```
Sub Sort(InData() As String, OutData() As String)
```

```
  OutData = InData       'Copia dei dati
  QuickSort OutData      'Ordinamento dei dati
End Sub
```

La procedura `Sort( )` ordina il vettore di stringhe `InData`. La procedura utilizza l'argomento `OutData` per restituire i dati ordinati senza alterare la matrice di partenza `InData`. Infatti, la procedura `Sort( )` prima copia la matrice `InData` nella matrice `OutData` e solo successivamente chiama la procedura `QuickSort( )`, che esegue l'ordinamento dei dati presenti in `OutData`.

Dichiarazione di matrici multidimensionali

Visual Basic consente di dichiarare matrici multidimensionali servendosi di indici multipli (uno per ciascuna dimensione). Il tipo più comune di matrice multidimensionale è quella composta da righe e colonne (in una disposizione di tipo tabellare) ed è, quindi, composta da due dimensioni. Un altro tipo di matrice multidimensionale abbastanza comune prevede tre dimensioni in una sorta di "cubo" nel quale i dati si sviluppano in altezza, in larghezza e in profondità (per esempio, una matrice di tabelle sovrapposte).

Quando si dichiara una matrice multidimensionale è necessario, innanzitutto, specificarne il nome, l'intervallo di indici e il tipo di dati degli elementi della matrice stessa.

La sintassi generale per dichiarare una matrice multidimensionale è la seguente.

```
Dim nomeMatrice([inizio1 To] Fine1, [inizio2 To] Fine2
[, ...]) As tipo
```

L'enunciato `Dim` specifica il nome della matrice, seguito dai rispettivi intervalli di indice (che specificano, nel contempo, il numero delle dimensioni: in sostanza, il numero degli intervalli indicati definisce la multidimensionalità della matrice) e, quindi, il tipo di dati che la matrice dovrà contenere. Si possono anche specificare il limite inferiore e superiore dei valori della matrice e se si omette l'indicazione di quello inferiore, tale limite dipenderà dal valore dell'enunciato `Option Base` tramite il quale si potrà impostare il limite inferiore di default a un valore pari a 0 oppure a 1. In mancanza dell'enunciato `Option Base`, il limite inferiore di una matrice viene, per default, impostato a 0.

Per maggiori informazioni circa le matrici multidimensionali, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

L'esempio seguente indica come dichiarare una matrice multidimensionale.

```
Option Base 1
Dim Giorni(1 To 7; 1 To 52) As Integer
Dim Anni(1980 To 2000, 12) As Long
Dim miaMatrice(10, 5, 2) As Double
```

La prima linea di questo codice dichiara la matrice bidimensionale `Giorni` con gli indici compresi tra 1 e 7 e da 1 a 52. Gli elementi della matrice memorizzano valori di tipo `Integer` e la matrice stessa sarà composta da 364 (7 x 52 = 364) elementi.

La seconda linea del codice dichiara la matrice bidimensionale `Anni` con un intervallo di indici da "1980" a "2000" e da "1" (default impostato dall'enunciato `Option Base`) a "12", Gli elementi della matrice saranno di tipo `Double` e la matrice stessa sarà composta da 252 (21 x 12 = 252) elementi.

L'ultima linea del codice definisce la matrice tridimensionale `miaMatrice` con l'intervallo di indici compreso tra "1" (default impostato dall'enunciato `Option Base`) a 10, 1 e 5 e da 1 a 2. Gli elementi della matrice saranno di tipo `Double` e la matrice stessa sarà composta da 100 (10 x 5 x 2 = 100) elementi.

Dichiarazione di matrici monodimensionali

Le matrici monodimensionali sono la forma più semplice di matrice e potrebbero essere paragonate a una sola riga o colonna di dati. Per esempio, si potrà usare una matrice monodimensionale per memorizzare i valori delle temperature massime di ogni giorno dell'anno.

Quando si dichiara una matrice monodimensionale si deve specificarne il nome, l'intervallo di indici e il tipo di dati ammessi.

La sintassi generale per dichiarare una matrice monodimensionale è la seguente.

```
Dim nomeMatrice([inizio To] fine) As tipo
```

L'enunciato `Dim` specifica il nome della matrice seguito, racchiuso fra parentesi tonde, dall'intervallo degli indici e quindi, dal tipo di dati che la matrice dovrà contenere. Si ha la possibilità di specificare il limite inferiore e superiore dell'intervallo degli indici e, in questo caso, il numero degli elementi della matrice corrisponderà a `fine-inizio+1`. Se si omette di indicare il limite inferiore, tale limite dipenderà dal valore dell'enunciato `Option Base` tramite il quale si può definire il limite inferiore di default impostandone il valo-

re a 0 o a 1. In mancanza dell'enunciato `Option Base`, il limite inferiore di una matrice verrà per default impostato a 0.

Per maggiori informazioni circa le matrici monodimensionali, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

L'esempio seguente indica come sia possibile dichiarare varie matrici monodimensionali.

```
Dim Giorni(1 To 7) As Integer
Dim Anni(1980 To 2000) As Long
Dim miaMatrice(10) As Double
```

La prima linea del codice dichiara la matrice `Giorni` con un intervallo di indici da 1 a 7. Gli elementi della matrice devono essere di tipo `Integer` e la matrice è composta da 7 elementi. La seconda linea del codice dichiara la matrice `Anni` con gli indici compresi tra 1980 e 2000. Gli elementi della matrice sono di tipo `Double` e la matrice è composta da 20 elementi. La terza linea del codice dichiara la matrice `miaMatrice` con un intervallo di indici che va da 0 (valore di default in mancanza dell'enunciato `Option Base`) a 10. Gli elementi della matrice saranno di tipo `Double` e la matrice stessa sarà composta da 11 elementi.

Visual Basic consente anche di dichiarare una matrice monodimensionale all'interno di una variabile di tipo `Variant`. Per farlo, occorre prima dichiarare una variabile di tipo `Variant` e quindi assegnarle la matrice di valori per mezzo della funzione `Array( )`. Gli elementi della matrice all'interno della variabile `Variant` possono essere utilizzati per mezzo di un indice, come quelli di una comune matrice.

L'esempio seguente indica come sia possibile dichiarare in questo modo una matrice monodimensionale.

```
Dim Numeri As Variant
Numeri = Array(1, 2, 3, 4)
For I = LBound(Numeri) To UBound(Numeri)
  Debug.Print Numeri(I)
Next I
```

L'esempio dichiara la variabile `Numeri`, di tipo `Variant`, e quindi vi inserisce la matrice di valori 1,2,3 e 4 per mezzo della funzione `Array( )`. Questo assegnamento fa si che Visual Basic consideri `Numeri` come una matrice.

Gli elementi presenti nella variabile `Numeri` vengono quindi visualizzati da un ciclo `For`. Si noti che l'elemento di partenza e quello di arrivo vengono ricavati per mezzo delle funzioni `LBound( )` e

`UBound( )`, che normalmente sono utilizzate solo per le matrici, non per i `Variant`. Stessa considerazione per quanto riguarda la lettura dei valori, eseguita dall'espressione `Numeri(I)`.

Matrici come argomenti di funzioni e procedure

Visual Basic consente alle funzioni e alle procedure di ammettere come argomenti una qualsiasi *matrice aperta*. Le matrici di questo tipo sono argomenti che specificano il tipo dell'elemento della matrice senza, tuttavia, specificare il *numero* degli elementi. Inoltre, in Visual Basic, le matrici aperte non necessitano della dichiarazione del numero delle dimensioni.

La sintassi generale per una matrice aperta è la seguente.

```
nomeMatrice( ) As tipo
```

`nomeMatrice` corrisponde al nome dell'argomento mentre tipo definisce il tipo di dati degli elementi della matrice.

Si eviti di passare matrici come valori per argomenti di matrice quando le dimensioni sono diverse. Per esempio, non si passi una matrice multidimensionale quando la routine prevede un argomento matrice monodimensionale.

Ricordare. Quando si dichiara un argomento matrice come `Variant`, si potranno anche passare matrici di tipo `Variant` per elaborare stringhe o valori numerici, poiché i passi richiesti sono esattamente gli stessi sia per le stringhe sia per i numeri. Un esempio classico, potrebbe essere l'ordinamento o la ricerca di dati.

L'esempio seguente mostra come sia possibile gestire un argomento matrice di tipo monodimensionale per ordinare una matrice di stringhe.

```
Sub Sort(Arr( ) As String, First As Integer, Last As
Integer)
  Dim I As Integer
  Dim J As Integer
  Dim OrdStr As String

  For I = Fisrt To Last - 1
    For J = I + 1 To Last
      If Arr(I) > Arr(J) Then
        OrdStr = Arr(I)
        Arr(I) = Arr(J)
        Arr(J) = OrdStr
      End If
```

```
    Next J
  Next I
End Sub
Private Sub Form_Load( )
  Dim Nomi(10) As String
  Dim I As Integer
  Dim Count As Integer

  ' assegna le stringe ad alcuni
  ' elementi della matrice
  Nomi(1) = "John"
  Nomi(2) = "Lara"
  Nomi(3) = "Emilio"
  Nomi(4) = "Marzia"
  Nomi(5) = "Joe"
  Count = 5
  ' visualizza la matrice non ordinata
  Debug.Print "Matrice non ordinata: "
  For I = 1 To Count
    Debug.Print Nomi(1)
  Next I
  ' ordina la matrice
  Sort Nomi, 1, Count
  ' visualizza la matrice ordinata
  Debug.Print "Matrice ordinata: "
  For I = 1 To Count
    Debug.Print Nomi(1)
  Next I
End Sub
```

Questo codice genererà, nella finestra Immediata il seguente risultato.

```
Matrice non ordinata:
John
Lara
Emilio
Marzia
Joe

Matrice ordinata:
Emilio
Joe
John
Lara
Marzia
```

Questo risultato riporta prima gli elementi di una matrice stringa non ordinati seguiti dagli elementi della stessa matrice elencati in

ordine alfabetico.La procedura `Sort( )` è destinata,quindi,a generare un ordinamento di una matrice di stringhe ed è associata ai seguenti argomenti.

- L'argomento `Arr` che corrisponde a una matrice aperta di stringhe.
- Gli argomenti `First` e `Last` di tipo `Integer` che rappresentano l'intervallo degli indici dei primi elementi della matrice da ordinare. Questi argomenti consentono di ordinare, quando necessario, una parte degli elementi della matrice.

La procedura `Sort( )` utilizza cicli `For` annidati che si occupano di riorganizzare gli elementi della matrice.

Questi cicli annidati confrontano gli elementi fra di loro contigui della matrice `Arr` per ordinarli servendosi di un metodo di ordinamento definito ***Bubble Sort***.

La procedura `Form_Load( )` dichiara la matrice di stringhe locale `Nomi` e le variabili `I`,`J` e `Count` di tipo `Integer` eseguendo,quindi le seguenti operazioni.

- Assegna i valori ai primi cinque elementi della matrice `Nomi`.
- Assegna alla variabile `Count` il valore 5.
- Visualizza, nella finestra Immediata, i primi cinque elementi della matrice `Nomi` in forma non ordinata.
- Ordina i primi cinque elementi della matrice non ordinata `Nomi`. Questa operazione si serve della procedura `Sort( )` e trasferisce i valori `Nomi`, `1` e `Count`, rispettivamente, agli argomenti `Arr`, `First` e `Last`.
- Visualizza, nella finestra Immediate, i primi cinque elementi della matrice `Nomi` in forma ordinata.

L'esempio seguente è una procedura il cui argomento corrisponde a un array di tipo bidimensionale (chiamato anche `matrice`).

```
Sub visMatr(Matr( ) As Double, Righe As Integer, Col As
Integer)
  Dim I As Integer
  Dim J As Integer

  For I = 1 To Righe
    For J = 1 To Col
      Debug.Print Mat(1, J); "  ";
    Next J
    Debug.Print
  Next I
End Sub
```

```
Private Sub Form_Load( )
  Dim Matrix(5, 5) As Double
  Dim I As Integer
  Dim J As Integer

  For I = 1 To 5
    For J = 1 To 5
      Matrix(I, J) = I + 10 * J
    Next J
  Next I
  visMatr Matrix, 5, 5
End Sub
```

Il risultato, nella finestra Immediata, di questo codice è il seguente.

```
11  21  31  41  51
12  22  32  42  52
13  23  33  43  53
14  24  34  44  54
15  25  35  45  55
```

Questo codice dichiara la procedura `visMatr( )` che prevede i seguenti argomenti.

- L'argomento `Mat` che corrisponde a una matrice aperta con elementi di tipo `Double`.
- Gli argomenti `Righe` e `Col` di tipo `Integer` che rappresentano il numero di righe e colonne da visualizzare. Questi argomenti consentono di visualizzare, quando necessario, solo una parte della matrice.

La procedura `visMatr( )` si serve di cicli `For` per visualizzare, nella finestra Immediata, gli elementi dell'argomento `Mat`. La routine visualizza gli elementi dell'argomento matrice disponendoli su righe.

La procedura `Form_Load( )` dichiara la matrice locale `Matrix` di tipo `Double` e le variabili `I` e `J` di tipo `Integer`. La procedura esegue quindi le seguenti operazioni.

- Assegna i valori agli elementi della matrice `Matrix`. Questa operazione si serve di cicli `For` e assegna, a ogni elemento della matrice, il valore (`I + 10 * J`). Questa espressione utilizza i valori del contatore del ciclo.
- Visualizza gli elementi della matrice `Matrix` nella finestra Immediate. Questa operazione richiama la procedura `visMatr()` e passa i valori `Matrix`, `5` e `5`, rispettivamente, agli argomenti `Mat`, `Righe` e `Col`.

Parte XI

Uso e uscita dai cicli

I cicli permettono alle applicazioni di iterare alcune operazioni fino a quando non viene soddisfatta una determinata condizione come, per esempio, l'avvio di conteggi sequenziali progressivi o regressivi. Un ciclo può prevedere di riportare il controllo all'inizio o alla fine di un programma oppure potrà prevedere un'uscita prematura dal programma stesso.

Argomenti trattati

- **Uso dei cicli per ripetere operazioni all'interno di un programma**
- **Le condizioni e i modi per uscire da un ciclo**
- **Avvio di un ciclo dalla fine o dall'inizio**

Ciclo Do Until

Un ciclo `Do Until` è di tipo condizionale, ovvero avvia un'iterazione di operazioni fino a quando una condizione predeterminata risulta *vera* ed esamina le condizioni *all'inizio* del ciclo stesso.

Vedi anche: in questa parte, la sezione che descrive i cicli `Do-Loop Until`.

La sintassi generale per l'uso di un ciclo `Do Until` è la seguente.

```
Do Until condizione
    enunciati
Loop
```

Questa sintassi prevede l'uso delle parole chiave `Do` e `Until` e la seconda viene immediatamente seguita dalla condizione da verificare. Se tale condizione risulta falsa già in fase di primo controllo, il ciclo non eseguirà gli `enunciati` immediatamente successivi. Infine, il ciclo viene concluso dalla parola chiave `Loop`.

Non si dimentichi di aggiornare il o i valori delle variabili utilizzate per verificare la condizione dell'enunciato del ciclo. In caso contrario la condizione non verrà mai modificata e il ciclo eseguirà un'iterazione senza fine.

Per maggiori informazioni sui cicli, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Viene ora proposto un esempio di uso di un ciclo `Do Until`.

```
Private Sub Form_Load( )
  Dim Somma As Double
  Dim Num As Integer
  ' inizializza le variabili
  Num = 1
  Somma = 0
  ' aggiunge numeri dispari
  ' compresi fra 1 e 99
  Do Until Num >= 100
    Somma = Somma + Num
    Num = Num + 2
  Loop
  Debug.Print Somma
End Sub
```

Il risultato di questo esempio visualizza, nella finestra Immediata il seguente risultato.

```
2500
```

2500 corrisponde alla somma di tutti i numeri dispari interi compresi tra 1 e 99. La procedura `Form_Load( )` dichiara le variabili `Somma` di tipo `Double` e `Num` di tipo `Integer` e le inizializza, rispettivamente, a 0 e 1. La procedura quindi avvia un ciclo `Do Until` per aggiungere alla variabile `Somma` valori interi dispari compresi fra 1 e 99. Il ciclo verifica la condizione `Num >= 100` per interrompere le proprie iterazioni quando il valore della variabile `Num` supera il 100. Il ciclo contiene due enunciati: il primo aggiunge il valore della variabile `Num` alla variabile `Somma` mentre il secondo aggiunge 2 alla variabile `Num` per memorizzare nella stessa il successivo valore dispari.

Ciclo Do While

Il ciclo `Do While` è di tipo condizionale e avvia un'iterazione *fintanto che* la condizione verificata risulta vera. Questo tipo di ciclo esamina la condizione all'*inizio* del ciclo stesso.

La sintassi generale di questo ciclo è la seguente.

```
Do While condizione
    enunciati
Loop
```

In questa sintassi appaiono le parole chiave `Do` e `While` e la seconda deve essere necessariamente seguita da una condizione da verificare. Se la condizione risulta falsa già in fase di prima verifica, il ciclo non esegue gli enunciati indicati. La parola chiave `Loop`, infine conclude l'iterazione.

Non si dimentichi di aggiornare il o i valori delle variabili utilizzate per verificare la condizione dell'enunciato del ciclo. In caso contrario la condizione non verrà mai modificata e il ciclo eseguirà un'iterazione senza fine.

Ed ecco un esempio per l'utilizzo di un ciclo `Do While`.

```
Private Sub Form_Load( )
  Dim Somma As Double
  Dim Num As Integer
  ' inizializza le variabili
  Num = 1
  Somma = 0
  ' aggiunge numeri dispari
  ' compresi tra 1 e 99
  Do While Num < 100
    Somma = Somma + Num
    Num = Num + 2
  Loop
```

```
  Debug.Print Somma
End Sub
```

Questo esempio visualizza, nella finestra **Immediata** il seguente valore:

```
2500
```

2500 corrisponde alla somma di tutti i numeri dispari compresi tra 1 e 99. La procedura `Form_Load( )` dichiara la variabile `Somma` di tipo `Double` e la variabile `Num` di tipo `Integer` e le inizializza, rispettivamente, a 0 e 1. La procedura quindi avvia un ciclo `Do While` per aggiungere alla variabile `Somma` i valori dispari interi compresi tra 1 e 99. Il ciclo verifica la condizione `Num < 100` ed esegue le iterazioni fino a che il valore della variabile `Num` sarà inferiore a 100. Il ciclo contiene due enunciati: il primo aggiunge il valore della variabile `Num` alla variabile `Somma` mentre il secondo aggiunge 2 al valore della variabile `Num` per memorizzare nella stessa il successivo numero dispari.

Ciclo Do-Loop Until

Il ciclo `Do-Loop Until` è di tipo condizionale e viene ripetuto fino a quando la condizione risulta vera. Questo tipo di ciclo esamina la condizione alla fine del ciclo.

Vedi anche: in questa parte, la sezione che descrive il ciclo `Do Until`.

La sintassi generale per l'uso di questo ciclo è la seguente.

```
Do
    enunciati

Loop Until condizione
```

Questa sintassi utilizza inizialmente la parola chiave `Do` seguita da uno o più enunciati. Il ciclo si conclude con le parole chiave `Loop Until` seguite, immediatamente dopo, dalla condizione da verificare. Se la condizione risulta falsa già alla prima verifica, il ciclo esegue una sola volta quanto richiesto dagli enunciati.

Non si dimentichi di aggiornare il o i valori delle variabili utilizzate per verificare la condizione dell'enunciato del ciclo. In caso contrario la condizione non verrà mai modificata e il ciclo eseguirà un'iterazione senza fine.

Viene ora proposto un esempio per l'uso del ciclo `Do-Loop Until`.

```
Private Sub Form_Load( )
  Dim Somma As Double
  Dim Num As Integer
  ' inizializza le variabili
  Num = 1
  Somma = 0
  ' aggiunge numeri dispari
  ' compresi tra 1 e 99
  Do
    Somma = Somma + Num
    Num = Num + 2
  Loop Until Num >= 100
  Debug.Print Somma
End Sub
```

Il risultato di questo esempio è la visualizzazione, nella finestra Immediate del seguente risultato.

```
2500
```

2500 corrisponde alla somma di tutti i numeri dispari compresi tra 1 e 99. La procedura `Form_Load( )` dichiara la variabile `Somma` di tipo `Double` e la variabile `Num` di tipo `Integer` e le inizializza, rispettivamente, a 0 e 1. La procedura, quindi, avvia un ciclo `Do-Loop Until` e aggiunge alla variabile `Somma` valori interi dispari compresi tra 1 e 99. Il ciclo verifica la condizione `Num >= 100` posta alla fine del ciclo stesso e interrompe le iterazioni quando il valore della variabile `Num` eccede 100. Il ciclo contiene due enunciati: il primo aggiunge al valore della variabile `Somma` il valore della variabile `Num` mentre il secondo aggiunge 2 al valore della variabile `Num` in modo da memorizzare nella stessa il successivo valore dispari.

Ciclo Do-Loop While

Il ciclo `Do-Loop While` è di tipo condizionale che esegue un'iterazione *mentre* una determinata condizione risulta vera. Questo tipo di ciclo esamina la condizione alla fine del ciclo.

La sintassi generale per l'uso di questo tipo di ciclo è la seguente.

```
Do
    enunciati

Loop While condizione
```

Questa sintassi utilizza la parola chiave `Do` iniziale seguita da uno o più enunciati. Il ciclo si conclude con le parole chiave `Loop While` seguite dalla condizione da verificare.

Se tale condizione risulta falsa già alla prima verifica, gli enunciati definiti dal ciclo vengono eseguiti una sola volta.

Non si dimentichi di aggiornare il o i valori delle variabili utilizzate per verificare la condizione dell'enunciato del ciclo. In caso contrario la condizione non verrà mai modificata e il ciclo eseguirà un'iterazione senza fine.

Viene ora proposto un esempio per l'uso del ciclo `Do-Loop While`.

```
Private Sub Form_Load( )
  Dim Somma As Double
  Dim Num As Integer
  ' inizializza le variabili
  Num = 1
  Somma = 0
  ' aggiunge numeri dispari
  ' compresi tra 1 e 99
  Do
    Somma = Somma + Num
    Num = Num + 2
  Loop While Num < 100
  Debug.Print Somma
End Sub
```

Il risultato di questo esempio è la visualizzazione, nella finestra Immediata del seguente risultato.

```
2500
```

2500 corrisponde alla somma di tutti i numeri dispari compresi tra 1 e 99. La procedura `Form_Load( )` dichiara la variabile `Somma` di tipo `Double` e la variabile `Num` di tipo `Integer` e le inizializza, rispettivamente, a 0 e 1. La procedura, quindi, avvia un ciclo `Do-Loop While` e aggiunge alla variabile `Somma` valori interi dispari compresi tra 1 e 99. Il ciclo verifica la condizione `Num < 100` posta alla fine del ciclo stesso ed esegue le iterazioni fintanto che il valore della variabile `Num` è inferiore a 100. Il ciclo contiene due enunciati: il primo aggiunge al valore della variabile `Somma` il valore della variabile `Num` mentre il secondo aggiunge 2 al valore della variabile `Num` in modo da memorizzare nella stessa il successivo valore dispari.

Exit Do

Visual Basic consente di uscire sempre da un ciclo di tipo "Do" utilizzando l'enunciato `Exit Do`. In un ciclo possono essere inseriti più enunciati `Exit Do` e, di solito, per valutare una condizione che esegue l'enunciato `Exit Do`, si utilizza un enunciato `If` nel quale

`Exit Do` trasferisce l'esecuzione del programma all'enunciato immediatamente successivo l'enunciato `Loop`.

La sintassi generale per l'enunciato `Exit Do` è la seguente.

```
Exit Do
```

Viene ora proposto un esempio pratico per l'uso dell'enunciato.

```
Dim I As Integer
Dim X As Double
I = 5
Do While I > = -5
  X = Sqr(I)  ' calcola
  Debug.Print X
  I = I - 1
  If I < 0 Then Exit Do
Loop
```

Questo esempio dichiara le due variabili I di tipo `Integer` e X di tipo `Double`. L'esempio inizializza la variabile I assegnandole il valore 5 e avvia un ciclo `Do While` per avviare le iterazioni fino a quando la variabile I è maggiore o uguale a –5. Il primo enunciato assegna alla variabile X il valore della radice quadrata della variabile I e visualizza il valore risultante nella finestra Immediata. Il terzo enunciato del ciclo diminuisce di 1 il valore della variabile I mentre l'ultimo enunciato del ciclo è un enunciato `If` che determina se il valore della variabile I è negativo. Se questa condizione è vera, il codice esegue l'enunciato `Exit Do` per uscire dal ciclo.

Exit For

Visual Basic consente di uscire da un ciclo di tipo "For", prima che le variabili di controllo del ciclo raggiungano il valore definito da `To`, tramite l'enunciato `Exit For`. In un ciclo si possono usare più enunciati `Exit For` e, di solito, per valutare la condizione che esegue un enunciato `Exit For` si utilizza un enunciato `If`. L'enunciato `Exit For` trasferisce l'esecuzione del programma all'enunciato immediatamente precedente la parola chiave `Next`.

La sintassi generale per l'uso di `Exit For` è la seguente.

```
Exit For
```

L'esempio qui proposto indica un modo per l'utilizzo di `Exit For`.

```
Dim I As Integer
  Dim X As Double
For I = 5 To -5 Step -1
  If I < 0 Then Exit For
```

```
  X = Sqr(I)   ' calcola
  Debug.Print X
Next I
```

L'esempio dichiara la variabile I di tipo `Integer` e la variabile X di tipo `Double` e contiene un ciclo `For` a conteggio regressivo che esegue un'iterazione fra i valori compresi tra 5 e –5. La variabile di controllo del ciclo è la variabile I.

Il primo enunciato del ciclo è un enunciato `If` che determina se la variabile di controllo del ciclo è negativa. Se questa condizione è vera, il codice esegue l'enunciato `Exit For` per uscire dal ciclo. In caso contrario il ciclo assegna la radice quadrata della variabile di controllo del ciclo alla variabile `X` e, quindi, visualizza nella finestra Immediata il valore di tale variabile. Il ciclo `For` esegue le proprie iterazioni da 5 a 0.

Ciclo For

Il ciclo `For` avvia un'iterazione per un numero predeterminato di volte. Questo ciclo si serve di una variabile speciale chiamata *variabile di controllo del ciclo* (o *contatore del ciclo*) che viene utilizzata per gestire il numero delle iterazioni del ciclo. L'enunciato del ciclo può usare il contatore nei seguenti modi.

- Come variabile che viene modificata a ogni iterazione.
- Come indice di una matrice alla quale il ciclo possa accedere.
- Come sistema di controllo delle iterazioni del ciclo.

La sintassi generale del ciclo `For` è la seguente.

```
For contatore = inizio To fine [Step passo]
  [   enunciati]
  [   Exit For
  [   enunciati]
Next [contatore]
```

`contatore` è la variabile di controllo del ciclo (o contatore del ciclo) e deve essere un valore numerico. Le parti `inizio` e `fine` corrispondono ai valori iniziale e finale del contatore del ciclo e specificano, rispettivamente, il valore iniziale e finale del ciclo stesso. Di solito, il valore di `inizio` dovrebbe essere inferiore o uguale a quello di `fine` e la clausola facoltativa `passo` definisce l'ammontare della modifica del valore del contatore del ciclo dopo ogni iterazione. L'incremento di default per il contatore del ciclo viene automaticamente impostato a 1 dopo ogni iterazione.

Quando il valore di `passo` è negativo,il ciclo eseguirà un conteggio di tipo regressivo e il valore di `inizio` dovrebbe, in questo caso, essere superiore a quello definito per `fine`. L'indicazione facoltativa di `Exit loop` consente di uscire prematuramente dal ciclo di iterazioni prima che il contatore abbia raggiunto il valore `fine`.

L'enunciato `Next` specifica la fine del ciclo `For` e se in questo enunciato si omette di indicare il contatore del ciclo,l'esecuzione del programma continuerà come se il contatore fosse stato indicato.

Nota. I cicli `For` sono ideali per l'elaborazione di elementi di una matrice. Le variabili di controllo del ciclo vengono utilizzate come indici per accedere alla matrice stessa.

Per maggiori informazioni sui cicli `For`,si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Qui di seguito viene ora fornito un esempio dell'uso di un ciclo `For`.

```
Private Sub Form_Load( )
  Dim I As Integer
  Dim Somma As Double
  ' inizializza Somma
  Somma = 0
  For I = 1 To 100 Step 2
    Somma = Somma + 1
  Next I
  Debug.Print Somma
End Sub
```

Il risultato di questo esempio è la visualizzazione, nella finestra Immediata, del seguente valore:

```
2500
```

2500 corrisponde alla somma di tutti i numeri dispari compresi tra 1 e 99. La procedura `Form_Load( )` dichiara la variabile `Somma` di tipo `Double` e la variabile `I` di tipo `Integer` assegnando quindi alla variabile `Somma` il valore 0. La procedura utilizza quindi un ciclo `For` per aggiungere alla variabile `Somma` un valore dispari compreso tra 1 e 99; il ciclo si serve quindi della variabile `I` utilizzandole come contatore e avvia le iterazioni da 1 a 100, con un incremento di 2. Ciò significa che la variabile `I` inizia con il valore 1 (il primo numero dispari nell'intervallo da 1 a 100) e a ogni iterazione viene modificata incrementandone il valore di 2. Il ciclo contiene un solo enunciato che aggiunge la variabile `Num` al valore della variabile `Somma`.

Ciclo For Each

Il ciclo `For Each` consente di ripetere uno o più enunciati per ogni elemento di una matrice o di una collezione di oggetti.

La sintassi generale del ciclo `For Each` è la seguente.

```
For Each elemento In gruppo
  [  enunciati]
  [  Exit For]
  [  enunciati]
Next [elemento]
```

`elemento` è una variabile utilizzata per avviare l'iterazione degli elementi di un gruppo o di una matrice. Nel caso di un gruppo, `elemento` può essere solo una variabile di tipo `Variant`, una variabile oggetto generica o una qualsiasi variabile oggetto specificata. Nel caso delle matrici, `elemento` può essere solo una variabile di tipo `Variant`. La voce `gruppo` corrisponde al nome di una collezione di oggetti o di una matrice (con l'eccezione di una matrice con tipo di dati definiti dall'utente). Il programma esegue gli enunciati del ciclo `For Each` solo nel caso in cui gruppo preveda la presenza di almeno un elemento.

L'enunciato `Exit For` consente di uscire prematuramente dal ciclo, tenendo presente che un ciclo può contenere più di un enunciato `Exit For`. Dopo la parola chiave `Next` si può evitare di indicare `elemento`.

Ed ecco un esempio dell'uso del ciclo `For Each`.

```
Private Sub Form_Load( )
  Dim Numeri(50) As Integer
  Dim I As Integer
  Dim Num As Variant

  ' assegna numeri casuali alla matrice
  For I = LBound(Numeri) To Ubound(Numeri)
    Numeri(I) = 1000 * Rnd( )
  Next I
  ' ricerca un valore che sia inferiore a 100
  For Each Num In Numeri
    If Num < 100 Then
      Debug.Print Num
    End If
  Next Num
End Sub
```

Il risultato di questo esempio visualizza, nella finestra Immediata, i 51 numeri casuali che sono inferiori a 100.

La procedura `Form_Load( )` dichiara la matrice `Numeri` composta da 50 elementi e dichiara la variabile `I` di tipo `Integer` e la variabile `Num` di tipo `Variant`. La procedura utilizza quindi un ciclo `For` assegnando agli elementi della matrice `Numeri` valori casuali (compresi nell'intervallo da 0 a 999). Viene poi utilizzato il ciclo `For Each` per eseguire un'iterazione sugli elementi della matrice `Numeri`. Il ciclo utilizza la variabile `Num` per accedere agli elementi della matrice. Lo stesso ciclo contiene l'enunciato `If-Then` che visualizza nella finestra Immediate il valore della variabile `Num`, sempre che questo sia inferiore a 100.

Per maggiori informazioni sui cicli `For Each`, si consulti il libro Visual Basic 6 For Dummies di Wallace Wang (Apogeo, 1998).

Terminologia Tecnica

ActiveX
Un nuovo nome per definire un controllo OLE.

Argomento
Dati trasferiti a una procedura o a una funzione.

Array
Variabile che memorizza valori multipli.

ASCII
Acronimo di *American Standard Code for Information Interchange.* Il codice numerico che rappresenta un carattere. Per esempio, il carattere "A" corrisponde al codice ASCII 65.

Barra dei menu
Finestra speciale che riporta un gruppo di menu. Ogni menu prevede una serie di propri comandi correlati ai quali l'utente potrà accedere aprendo il menu facendo clic sul suo nome oppure utilizzando un tasto acceleratore.

Bitmap
Formato per immagini grafiche che consiste in una serie di bit che traccino una "mappa" dell'immagine stessa.

Booleano
Sinonimo di logico. Il termine proviene dal nome del matematico George Boole. I valori booleani possono essere "vero" o "falso".

Chiamata di funzione
Porzione di codice che richiama una funzione passando valori agli argomenti della stessa.

Ciclo
Vedi Loop.

Classe
Tipo speciale di tipo di dati definito dall'utente che rappresenta gli attributi e le operazioni di una categoria di oggetti. Una classe, di solito, contiene elementi dati e procedure (vale a dire, procedure e funzioni).

Comando di menu
Comando accessibile dalla barra dei menu.

Concatenazione
Combinazione di due o più stringhe.

Condizione
Espressione logica che può assumere i valori "vero" e "falso".

Contatore loop
Variabile che controlla il numero di iterazioni in un loop. Sinonimo di variabile di controllo del loop.

Controllo
Componente Visual Basic al quale sono associate proprietà e che risponde agli eventi. Di solito, un controllo è visibile anche se, a volte, può essere definito come nascosto.

Convenzione standard per i nomi
Convenzione utilizzata per assegnare un nome agli elementi di un linguaggio di programmazione. La convenzione standard di Visual Basic richiede che un nome inizi sempre con un carattere alfabetico seguito da altri caratteri alfanumerici o da caratteri "". I nomi di Visual Basic non riconoscono l'uso di caratteri maiuscoli o minuscoli.

Costante
Identificatore (vale a dire un nome che deve seguire le convenzioni per l'assegnazione dei nomi in Visual Basic) associato con un valore che rimane invariato nel corso dell'esecuzione di un programma.

Design time
Si riferisce a un applicativo Visual Basic in fase di progettazione.

DHTML

HTML (HyperText Markup Language) Dinamico è una versione avanzata dell'HTML che supporta gli oggetti dinamici.

Elemento dati

Membro di una classe o struttura che memorizza un attributo o un valore di un oggetto.

Elemento della matrice

Un singolo valore di una matrice, al quale si accede utilizzando uno o più indici.

Enumeratore

Nome (associato a un valore fisso) che viene dichiarato in un tipo di dati enumerato.

Enumerazione

Il processo in base al quale si crea un tipo di dati enumerato.

Enunciato If

Enunciato in base al quale vengono prese delle decisioni e che fornisce alternative singole o multiple.

Esecuzione

Fa riferimento al momento nel quale si avvia un'applicazione Visual Basic.

Espressione

Serie di operatori e valori che generano un unico risultato.

Etichetta (Label)

Sinonimo di controllo di testo statico.

Evento

Azione (come, per esempio, il clic del mouse) al quale un controllo deve rispondere.

File .BAS.

Tipo di file nel quale vengono memorizzati i moduli Visual Basic.

File .BMP

Tipo di file destinato alla memorizzazione di immagini grafiche bitmap.

File .CLS

Tipo di file che memorizza i moduli di classe di Visual Basic.

File .FRM

File destinati a contenere i form Visual Basic.

File .HPJ

Tipo di file che memorizza le informazioni per la gestione dei file del sistema di guida.

File - VBP

Tipo di file che memorizza i progetti Visual Basic.

File Modulo

File contenente una libreria di procedure, funzioni, costanti globali, variabili e matrici. Un modulo principale o altri moduli possono condividere parti pubbliche di un file modulo.

Finestra di dialogo "Common dialog"

Una finestra di dialogo fornita da Windows 9x per l'esecuzione di alcune operazioni comuni a tutte le applicazioni. Le finestre di dialogo "Common dialog" sono, per esempio, quelle per l'apertura o il salvataggio dei file, per la modifica del tipo di carattere, per la stampa dei documenti e per l'accesso al sistema di guida.

Finestra Immediata (Finestra di debug)

Chiamata a volte *Finestra di debug* corrispondente all'oggetto Debug di Visual Basic. Di solito per visualizzare elementi in questa finestra, si usa il comando Debug.Print.

Finestra Watch (Espressioni di Controllo)

Finestra che consente di osservare i valori correnti delle variabili di un programma Visual Basic.

Form

Finestra speciale nella quale si possono disegnare i vari controlli. I moduli sono l'elemento centrale per la creazione di finestre per l'interfaccia utente delle applicazioni Visual Basic.

Funzione
Porzione di codice che esegue operazioni e restituisce un risultato.

Funzione ricorsiva
Funzione che richiama se stessa per generare un valore. Una funzione di questo tipo deve interrompere la propria chiamata quando una determinata condizione risulta vera. Ciò eviterà di avviare una chiamata infinita di funzione. Quello che segue è un esempio di funzione ricorsiva che calcola un numero fattoriale.

Si osservi che la funzione interrompe la propria ricorsività quando il valore dell'argomento è inferiore a 1

```
Function Fattoriale (N As Integer)
  If N > 1 Then
     Fattoriale = Fattoriale (N - 1)
  Else
     Fattoriale = 1
  End If
End Function
```

Gestore di evento
Procedura che risponde a un evento, quale, per esempio, un clic del mouse.

Identificatore
Nome di una componente di un programma come, per esempio, il nome di una costante o di una variabile.

Iterare
Eseguire un'operazione ripetutamente. Visual Basic utilizza i cicli per eseguire operazioni iterative.

Label
Vedi: Etichetta.

Letterale
Indica elementi di tipo letterale (sotto forma di stringa) quali, per esempio "Io", "10" o "Giovanni".

Loop
Parte di un programma che consente di ripetere una o più volte (iterare) una o più operazioni.

Loop Do Until

Un loop che esegue iterazioni fino a quando una condizione risulterà vera.

Loop Do While

Loop che esegue iterazioni mentre una condizione risulta vera.

Loop For

Loop che, di solito, esegue le iterazioni per un numero fisso di volte.

Matrice (Array)

Tipo di variabile speciale che memorizza più valori.

MDI

Acronimo di *Multiple Document Interface.* MDI si riferisce a un tipo di applicazione che consente di aprire e operare contemporaneamente su più documenti. Microsoft Word è un esempio di applicazione MDI.

Membro statico

Variabile locale che mantiene invariato il proprio valore tra una chiamata di procedura e l'altra.

Menu

Insieme di comandi che appare sulla Barra dei menu.

Metodo

Procedura o funzione speciale associata a un modulo, controllo od oggetto.

Modale

Stato di una finestra di dialogo che richiede che l'utente chiuda la finestra di dialogo prima di accedere a un'altra finestra dello stesso programma. Opposto di "Non modale".

Modulo classe

Modulo che definisce e implementa una classe Visual Basic.

Non Modale

Stato di una finestra di dialogo che consente all'utente di accedere ad altre finestre della stessa applicazione anche nel caso in cui un'altra finestra non modale risulta aperta. Opposto di "Modale".

Numero a virgola mobile

Numero composto da una parte intera e da una parte frazionaria. Il numero 10,34 è un esempio di numero a virgola mobile, ovvero 10 e 34 centesimi.

ODBC

Acronimo di *Open Data Base Connectivity*, tecnologia per l'accesso al motori database.

Oggetto

Istanza di una classe. Per esempio, un particolare modello di computer IBM è un oggetto e un'istanza della classe dei computer IBM.

OLE

Acronimo di *Object Linking and Embedding*. Si tratta di una funzione di programmazione che è in grado di riconoscere i controlli ActiveX e consente di incorporare o collegare altri documenti all'interno di un documento contenitore.

Operatore

Simbolo o nome che utilizza un valore e genera un risultato. Per esempio, il simbolo "+" corrisponde all'operatore per la somma. Il nome Mod corrisponde all'operatore "modulo" per la divisione di valori interi con resto.

Option Base

Enunciato che imposta il valore minimo di default di un indice di una matrice. L'indice minimo di default può essere 0 oppure 1. Visual Basic imposta per default Option Base al valore 0.

Presa di decisioni

Capacità di una procedura di esaminare una condizione ed eseguire un'azione in base al fatto che la condizione sia o meno vera. Per esempio, un programma di questo tipo può essere in grado di reperire un file, aprirlo e leggerne i dati e, nel caso in cui il file non possa essere reperito, visualizzare un messaggio di errore che indichi che il programma non è stato in grado di trovare il file.

Private

Tipo di elemento dati o procedura alla quale si può accedere solo dall'interno del modulo o della classe utilizzato per la dichiarazione dell'elemento. Opposto di "Public".

Procedura

Termine comunemente utilizzato per indicare una subroutine o una funzione.

Progettazione

Si riferisce alla fase nella quale Visual Basic non viene utilizzato e descrive il momento in cui si progetta un form.

Proprietà

Attributo di un controllo o di un oggetto.

Public

Tipo di elemento dati o di procedura alla quale si può accedere anche da altri moduli o classi. Opposto di "Private".

Pulsante di opzione

Controllo di Visual Basic composto da un'etichetta e da un elemento circolare. Quando si seleziona un pulsante di opzione, nell'elemento circolare viene riportato un pallino nero. L'assenza di un pallino nero all'interno di un pulsante di opzione indica che l'opzione stessa non è attiva.

Routine

Termine comunemente utilizzato per definire una procedura o una funzione.

Run time

Si riferisce a un applicativo Visual Basic in esecuzione.

Select Case

Enunciato che consente di prendere in considerazione più alternative.

SQL

Acronimo di *Structured Query Language*, linguaggio di interrogazione per i database.

Stringa

Una serie di caratteri testuali.

Stringa a lunghezza fissa

Stringa che può memorizzare solo fino a un numero specificato di caratteri.

Stringa a lunghezza variabile

Stringa che può memorizzare un numero di caratteri variabile.

Struttura

Tipo di dati definito dall'utente che dichiara uno o più elementi.

Sub

Parola chiave utilizzata per dichiarare una procedura.

Subroutine

Porzione di codice che esegue un'operazione all'interno di un codice globale.Subroutine viene anche utilizzato per indicare funzioni o procedure.

Tasto acceleratore (Hot Key)

Carattere che richiama un menu o un comando di menu, di solito prevede l'associazione al tasto Alt.

Tipo di dati

Categoria di informazioni, come numeri interi, caratteri, stringhe e numeri a virgola mobile o di tipo personalizzato.

Tipo enumerato

Tipo di dati definito dall'utente che contiene un elenco di nomi chiamati enumeratori. Ogni enumeratore corrisponde a una costante associata a un valore, il tipo enumerato consente di dichiarare una serie di costanti che descrivono i vari valori fissi o gli stati di un oggetto. Per esempio, si può dichiarare un tipo enumerato che definisca i giorni della settimana.

Twip

Unità di misura grafica. 1440 twip corrispondono a un pollice (25,4 mm) e 567 twip corrispondono a 1 cm.

Valore dell'argomento

Valore fornito all'argomento di una procedura o di una funzione quando la routine chiama questa procedura o funzione.

Valore dell'argomento di default

Un valore di default associato a un argomento. Se si omette un valore specifico per un determinato argomento che prevede l'applicazione di un valore di default, il computer automaticamente utilizza il valore di default.

Variabile

Identificatore (nome soggetto alle convenzioni per l'assegnazione dei nomi di Visual Basic) associato a un valore che può essere modificato nel corso dell'esecuzione di un programma.

Variabile di controllo del loop

Variabile che controlla il numero di iterazioni di un loop. Sinonimo di contatore Loop.

Variabile dinamica

Variabile, struttura od oggetto che viene creato in fase di esecuzione utilizzando l'operatore New.

Indice analitico

A

B

C

E

F

H

I

L

M

N

O

U

V

WINDOWS E APPLICATIVI

88-7303-450-0	I segreti di Windows 98	L. 98.000
88-7303-437-3	Windows 98 Guida completa	L. 69.000
88-7303-447-0	Windows 98 For Dummies	L. 36.000
88-7303-459-4	Windows 98 For Dummies Espresso	L. 19.000
88-7303-436-5	Windows 98 flash	L. 16.000
88-7303-461-6	Windows 98 Tutto&Oltre	L. 88.000
88-7303-451-9	Windows 98 Installazione e configurazione	L. 88.000
88-7303-438-1	I segreti della programmazione in Windows 98	L. 98.000
88-7303-460-8	L'API di Windows 98 For Dummies Espresso	L. 19.000
88-7303-398-9	Windows 98 Anteprima	L. 19.000
88-7303-479-9	Programmare in Windows 98/NT Tutto&Oltre	L. 88.000
88-7303-471-3	Introduzione a Windows NT 5	L. 45.000
88-7303-346-6	Windows 95 For Dummies Espresso	L. 19.000
88-7303-377-6	Windows 95 flash Seconda edizione	L. 16.000
88-7303-427-6	Windows 95 per tutti	L. 32.000
88-7303-298-2	I segreti di Windows NT Server 4	L. 78.000
88-7303-141-2	I segreti di Windows 95	L. 92.500
88-7303-340-7	Windows NT For Dummies	L. 32.000
88-7303-341-5	Windows 95 For Dummies	L. 32.000
88-7303-345-8	Windows NT For Dummies Espresso	L. 19.000
88-7303-302-4	Windows NT Workstation 4.0 flash	L. 16.000
88-7303-157-9	Guida a Windows 95	L. 39.000
88-7303-138-2	Windows 95 Guida pratica	L. 28.000
88-7303-139-0	Windows 95 Grande guida	L. 65.000
88-7303-367-9	Office 97 For Dummies	L. 32.000
88-7303-328-8	Office 97 Autoistruzione	L. 42.000
88-7303-354-7	Office 97 per Windows For Dummies Espresso	L. 19.000
88-7303-323-7	Word 97 flash	L. 16.000
88-7303-342-3	Word 97 For Dummies	L. 32.000
88-7303-347-4	Word 97 For Dummies Espresso	L. 19.000
88-7303-343-1	Excel 97 For Dummies	L. 32.000

88-7303-348-2	Excel 97 For Dummies Espresso	L. 19.000
88-7303-324-5	Excel 97 flash	L. 16.000
88-7303-339-3	I segreti di Excel 97	L. 78.000
88-7303-411-X	Laboratorio di Excel 97	L. 24.000
88-7303-174-9	Excel 95 flash	L. 16.000
88-7303-337-7	Excel 95 per tutti	L. 28.000
88-7303-169-2	Word 95 flash	L. 16.000
88-7303-336-9	Word 95 per tutti	L. 28.000
88-7303-199-4	PowerPoint 95 flash	L. 16.000
88-7303-178-1	Access 95 flash	L. 16.000
88-7303-170-6	I segreti di Word per Windows 95	L. 78.000
88-7303-161-7	I segreti di Excel per Windows 95	L. 78.000
88-7303-349-0	Access 97 For Dummies Espresso	L. 19.000
88-7303-344-X	Access 97 For Dummies	L. 32.000
88-7303-412-8	Laboratorio di Access 97	L. 28.000
88-7303-325-3	Access 97 flash	L. 16.000
88-7303-184-6	Access 95 per tutti	L. 28.000
88-7303-163-3	Works per Windows 95	L. 49.000
88-7303-326-1	PowerPoint 97 flash	L. 16.000
88-7303-198-6	Lotus Notes 4 flash	L. 16.000
88-7303-353-9	Outlook 97 For Dummies Espresso	L. 19.000
88-7303-449-7	Outlook 98 flash	L. 16.000
88-7303-095-5	Windows 3.1, Word 6, Excel 5, Access 2	L. 48.000
88-7303-038-6	Windows 3.1 Autoistruzione	L. 48.000
88-7303-048-3	Windows per tutti quelli che	L. 25.000
88-7303-066-1	Winword per tutti (per Windows 3.1)	L. 28.000
88-7303-078-5	Word 6 Autoistruzione	L. 35.000
88-7303-041-6	Il wordprocessing per tutti	L. 25.000
88-7303-107-2	dBase 5 per Windows	L. 28.000
88-7303-086-6	Access 2 Grande guida	L. 48.000
88-7303-037-8	Paradox per Windows Grande guida	L. 85.000
88-7303-083-1	Excel 5 Autoistruzione	L. 35.000

88-7303-161-7	I segreti di Excel per Windows 95 .... L. 78.000
88-7303-327-X	Outlook 97 flash .... L. 16.000

INTERNET

88-7303-407-1	Internet per tutti Terza edizione .... L. 32.000
88-7303-439-X	Internet For Dummies Quinta edizione .... L. 36.000
88-7303-356-3	Internet For Dummies Espresso .... L. 19.000
88-7303-422-5	Internet e Web flash .... L. 16.000
88-7303-152-8	Il mio server Web .... L. 65.000
88-7303-147-1	I segreti del World Wide Web .... L. 78.000
88-7303-140-4	Eudora, la posta elettronica via Internet .... L. 28.000
88-7303-314-8	Eudora flash .... L. 16.000
88-7303-206-0	Il modem per tutti .... L. 28.000
88-7303-120-X	Internet per le aziende .... L. 48.000
88-7303-202-8	Intranet flash .... L. 16.000
88-7303-210-9	Telefonare con Internet .... L. 36.000
88-7303-151-X	Il nuovo Navigare con Internet .... L. 58.000
88-7303-386-5	Internet Explorer 4 For Dummies .... L. 32.000
88-7303-383-0	Internet Explorer 4 For Dummies Espresso .... L. 19.000
88-7303-387-3	Netscape Communicator 4 For Dummies .... L. 32.000
88-7303-382-2	Netscape Communicator 4 F. D. Espresso .... L. 19.000
88-7303-408-X	Internet Explorer 4 flash .... L. 16.000
88-7303-376-8	Netscape Communicator 4 flash .... L. 16.000
88-7303-319-9	Inglese per Internet flash .... L. 24.000
88-7303-477-2	Ricerche online For Dummies .... L. 36.000
88-7303-180-3	Netscape flash .... L. 16.000

HTML

88-7303-394-6	HTML 4 .... L. 48.000
88-7303-393-8	HTML 4 Tutto&Oltre .... L. 88.000
88-7303-409-8	FrontPage 98 Guida completa .... L. 59.000

88-7303-365-2	HTML 4 For Dummies	L. 32.000
88-7303-355-5	HTML 4 For Dummies Espresso	L. 19.000
88-7303-392-X	HTML 4 flash	L. 16.000
88-7303-363-6	Creare pagine Web For Dummies	L. 32.000
88-7303-446-2	Costruire un sito Web For Dummies	L. 39.500
88-7303-445-4	XML For Dummies	L. 39.500
88-7303-188-9	PERL Guida pratica	L. 36.000
88-7303-329-6	Guida a FrontPage 97	L. 32.000
88-7303-307-5	HTML 3.2 Guida completa	L. 65.000
88-7303-196-X	HTML flash	L. 16.000
88-7303-320-2	HTML 3.2 Nuova edizione	L. 38.000
88-7303-153-6	VRML	L. 55.000
88-7303-352-0	Marimba Castanet la guida ufficiale	L. 46.000
88-7303-391-1	HTML dinamico Guida completa	L. 59.000

JAVA

88-7303-305-9	Programmare in JavaScript	L. 42.000
88-7303-316-4	Programmare in Visual J++	L. 58.000
88-7303-350-4	Java Restaurant	L. 28.000
88-7303-357-1	JavaScript For Dummies Espresso	L. 19.000
88-7303-423-3	Java For Dummies	L. 39.500
88-7303-466-7	Java 1.2 Tutto&Oltre	L. 88.000
88-7303-465-9	Java 1.2 Guida completa	L. 69.000
88-7303-197-8	Java flash	L. 16.000
88-7303-351-2	Java 1.1 Guida completa	L. 69.000
88-7303-189-7	Java 1.0 Guida completa	L. 64.000
88-7303-375-X	Java 1.1 Tutto&Oltre	L. 88.000

GRAFICA, CAD E DTP

88-7303-410-1	AutoCAD 14 Guida completa	L. 98.000
88-7303-360-1	AutoCAD 14 Guida all'uso	L. 59.000
88-7303-414-4	AutoCAD 14 For Dummies	L. 32.000

88-7303-318-0	I segreti di AutoCAD 13	L. 78.000
88-7303-415-2	AutoCAD 14 For Dummies Espresso	L. 19.000
88-7303-044-0	AutoCAD 12 Comandi	L. 58.000
88-7303-400-4	3D Studio MAX 2 Guida completa	L. 69.000
88-7303-380-6	3D Studio MAX per il professionista	L. 48.000
88-7303-399-7	LightWave 3D 5.5 Guida completa	L. 88.000
88-7303-300-8	3D Studio MAX Guida completa	L. 65.000
88-7303-321-0	LightWave Guida completa	L. 56.000
88-7303-135-8	3D Studio 4	L. 58.000
88-7303-072-6	Guide Apogeo: 3D Studio 3	L. 48.000
88-7303-458-6	Photoshop 5 For Dummies	L. 36.000
88-7303-455-1	Photoshop 5 Guida completa	L. 69.000
88-7303-473-X	Photoshop 5	L. 98.000
88-7303-331-8	Photoshop 4 Guida completa	L. 59.000
88-7303-368-7	Photoshop 4 per Windows For Dummies	L. 32.000
88-7303-101-3	Photoshop 3 per Macintosh e Windows	L. 58.000
88-7303-068-8	Guide Apogeo: Photoshop 2.5	L. 28.000
88-7303-430-6	CorelDRAW! 8 For Dummies	L. 36.000
88-7303-440-3	CorelDRAW! 8 Guida completa	L. 59.000
88-7303-444-6	QuarkXPress Versione 4 per Mac e PC	L. 49.000
88-7303-194-3	Lavorare con Adobe Illustrator	L. 36.000
88-7303-089-0	CorelDRAW! 5	L. 68.000
88-7303-052-1	PageMaker 5	L. 53.000
88-7303-082-3	QuarXPress 3.1	L. 48.000
88-7303-154-4	Adobe Acrobat per Macintosh e Windows	L. 30.000
88-7303-060-2	Il desktop publishing per tutti quelli che	L. 25.000

PROGRAMMAZIONE

88-7303-462-4	Visual Basic 6 Tutto&Oltre	L. 88.000
88-7303-456-X	Visual Basic 6 Guida completa	L. 69.000
88-7303-469-1	Visual Basic 6 For Dummies	L. 39.500

88-7303-304-0	Visual Basic flash	L. 16.000
88-7303-476-4	Visual Basic 6 For Dummies Espresso	L. 19.000
88-7303-381-4	Visual Basic 5 For Dummies Espresso	L. 19.000
88-7303-463-2	I segreti di Visual Basic 6	L. 88.000
88-7303-470-5	Visual Basic - La programmazione dei database	L. 59.000
88-7303-402-0	A scuola di Visual Basic 5 For Dummies	L. 39.500
88-7303-338-5	Visual Basic 5 Guida completa	L. 69.000
88-7303-369-5	Visual Basic 5 For Dummies	L. 32.000
88-7303-155-2	Programmare in Visual Basic 4	L. 64.000
88-7303-164-1	Visual Basic 4 Guida pratica	L. 29.000
88-7303-098-X	SuperKit Visual Basic	L. 28.000
88-7303-071-8	Programare con Visual Basic 3	L. 58.000
88-7303-084-X	Borland C++ 4 e 4.5	L. 58.000
88-7303-008-4	Programmare in C senza errori	L. 23.000
88-7303-371-7	Borland C++ Builder La guida ufficiale	L. 69.000
88-7303-333-4	I segreti di Visual Basic 5	L. 78.000
88-7303-454-3	Borland Delphi 4 Guida completa	L. 88.000
88-7303-421-7	Borland C++ Builder 3 Guida completa	L. 69.000
88-7303-474-8	Programmare con Delphi 4	L. 59.000
88-7303-361-X	Programmare in C	L. 64.000
88-7303-315-6	Programmare in C++	L. 59.000
88-7303-468-3	Visual C++ 6 For Dummies	L. 39.500
88-7303-464-0	Visual C++ 6 Guida completa	L. 69.000
88-7303-467-5	Visual C++ 6 Tutto&Oltre	L. 88.000
88-7303-475-6	Visual C++ 6 For Dummies Espresso	L. 19.000
88-7303-401-2	A scuola di C++ For Dummies	L. 39.500
88-7303-370-9	Visual C++ 5 For Dummies	L. 32.000
88-7303-374-1	Visual C++ 5 Guida completa	L. 69.000
88-7303-306-7	Programmare in Visual C++ 4.2	L. 74.000
88-7303-372-5	Borland Delphi 3 La guida ufficiale	L. 59.000
88-7303-201-X	Delphi 2	L. 42.000

SISTEMI OPERATIVI

88-7303-397-0	MAC per tutti Nuova edizione	L. 32.000
88-7303-384-9	Macintosh (MAC OS 8) For Dummies	L. 32.000
88-7303-429-2	Computer in rete For Dummies	L. 36.000
88-7303-204-4	Il computer per tutti Nuova edizione	L. 28.000
88-7303-301-6	Computer flash	L. 16.000
88-7303-200-1	Informatica Nuova edizione	L. 58.000
88-7303-385-7	PC For Dummies Per DOS e Windows	L. 32.000
88-7303-364-4	DOS For Dummies Per Windows 95	L. 32.000
88-7303-185-4	Il DOS per tutti	L. 28.000
88-7303-418-7	UNIX For Dummies	L. 36.000
88-7303-416-0	Linux For Dummies	L. 39.500
88-7303-435-7	Red Hat Linux Tutto&Oltre	L. 88.000
88-7303-419-5	UNIX For Dummies Espresso	L. 19.000
88-7303-417-9	Linux For Dummies Espresso	L. 19.000
88-7303-191-9	I segreti di Linux	L. 69.000
88-7303-299-0	I segreti di UNIX	L. 74.000
88-7303-317-2	Linux HowTo	L. 65.000
88-7303-303-2	UNIX flash	L. 16.000
88-85146-00-7	Lavorare con UNIX	L. 58.000

NUOVE FRONTIERE

88-7303-358-X	Ingegneria economica del software	L. 34.000
88-7303-478-0	Introduzione alla Matematica discreta	L. 40.000
88-7303-413-6	Sistemica	L. 24.000
88-7303-432-2	Atti del primo congresso italiano di Sistemica	L. 60.000
88-7303-119-6	Vita artificiale	L. 28.000
88-7303-002-5	Reti neurali artificiali	L. 25.000
88-7303-099-8	Intelligenza e vita	L. 28.000

GUIDE AGLI ESAMI DI CERTIFICAZIONE

88-7303-441-1	Nozioni fondamentali delle reti	L. 49.000
88-7303-443-8	NT Server 4	L. 49.000
88-7303-457-8	NT Server 4 Enterprise	L. 49.000
88-7303-472-1	Windows NT Workstation	L. 49.000
88-7303-442-X	TCP/IP	L. 49.000

CONNESSIONI

88-7303-172-2	Telematica per la pace	L. 26.000
88-7303-208-7	Sesto potere	L. 32.000
88-7303-183-8	Il telefonino	L. 24.000
88-7303-332-6	Incontri virtuali	L. 24.000
88-7303-207-9	La vita sullo schermo	L. 46.000
88-7303-359-8	Spaghetti hacker	L. 30.000
88-7303-395-4	Internet per la didattica	L. 34.000
88-7303-448-9	Internet per chi studia nuova edizione	L. 28.000
88-7303-453-5	Computer e scuola	L. 24.000
88-7303-335-0	Fare marketing con Internet	L. 28.000
88-7303-452-7	Trovare lavoro con Internet	L. 24.000
88-7303-396-2	Gens electrica	L. 28.000
88-7303-431-4	Ricerche bibliografiche in Internet	L. 24.000

MANUALI PRATICI

88-7303-378-4	Cucina facile For Dummies	L. 32.000
88-7303-379-2	Cucina dietetica For Dummies	L. 32.000
88-7303-390-3	Tenersi in forma For Dummies	L. 32.000
88-7303-434-9	Il mio cane For Dummies	L. 32.000
88-7303-433-0	Il mio gatto For Dummies	L. 32.000
88-7303-388-1	Musica classica For Dummies	L. 39.500
88-7303-389-X	Opera lirica For Dummies	L. 39.500

Finalmente un libro semplice, chiaro e divertente per capire Visual Basic 6

I piccoli manuali per comprendere l'informatica

Una guida di facile consultazione, concisa e affidabile alla programmazione in Visual Basic 6. Tutte le informazioni necessarie per lo sviluppo di applicazioni Windows riunite in uno strumento di riferimento chiaro e completo.

- Fondamenti di Visual Basic: controlli, proprietà e menu.
- Creazione di classi e tipi di dati.
- Utilizzo di finestre di dialogo, funzioni e procedure.
- Risposte chiare a tutte le domande sul debug e la gestione degli errori.
- Suggerimenti per l'utilizzo di stringhe, form, array e cicli.
- Integrazione di Visual Basic con gli altri programmi di Visual Studio.

Fatti guidare dalle icone

Indossare guanti protettivi e procedere con cautela

Una procedura diversa dal solito

Un suggerimento per eseguire un'operazione alternativa

I libri della collana *For Dummies* e *For Dummies Espresso* sono disponibili in libreria e nei computer shop. Per informazioni richiedete il catalogo Apogeo o visitate il sito: **http://www.apogeonline.com**

www.apogeonline.com

ISBN 88-7303-476-4

L. 19.000